HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELORARBEIT

Herr
Falco Greß

Konzept, Entwurf und prototypische Implementierung eines Kamerasystems für 2D und 2.5D Sidescroller Games zur einfachen Integrierung in eine aktuelle Game Engine

2015

Fakultät: Mathematik/Naturwissenschaften/Informatik

BACHELORARBEIT

Konzept, Entwurf und prototypische Implementierung eines Kamerasystems für 2D und 2.5D Sidescroller Games zur einfachen Integrierung in eine aktuelle Game Engine

Autor:
Herr Falco Greß

Studiengang:
Medieninformatik & Interaktives Entertainment

Seminargruppe:
MI11w1-b

Erstprüfer:
Prof. Dr.-Ing. Wilfried Schubert

Zweitprüfer:
Dipl.-Inf. Knut Altroggen

Einreichung:
Mittweida, 11.12.2015

Faculty of Mathematics/Physics/Computer Science

BACHELOR THESIS

Concept, design and prototypical implementation of a camera system for 2D and 2.5D side-scrolling games for an easy integration into a current game engine

author:
Mr. Falco Greß

course of studies:
Media Informatics and Interactive Entertainment

seminar group:
MI11w1-B

first examiner:
Prof. Dr.-Ing. Wilfried Schubert

second examiner:
Dipl.-Inf. Knut Altroggen

submission:
Mittweida, 11th of December 2015

Bibliografische Angaben

Greß, Falco:

Konzept, Entwurf und prototy-pische Implementierung eines Kamerasystems für 2D und 2.5D Sidescroller Games zur einfachen Integrierung in eine aktuelle Game Engine

Concept, design and prototypical implementation of a camera sys-tem for 2D and 2.5D side-scrolling games for an easy integration into a current game engine

2015 – 66 Seiten

Mittweida, Hochschule Mittweida, University of Applied Sciences,
Fakultät Mathematik/Naturwissenschaften/Informatik, Bachelorarbeit, 2015

Abstract

In der vorliegenden Arbeit wird anhand der Entwicklung einer Software mit den Teilschritten Konzeption, Design und Prototypentwicklung ein Einblick in das Forschungsfeld der Kameras in Videospielen zu gegeben. Dabei werden die Herausforderungen für Kameras in Sidescrollern erörtert. Anschließend werden verschiedene Sidescroller auf die Auswirkungen der Kamera auf das jeweilige Spiel hin untersucht. Im praktischen Teil der Arbeit wird zunächst auf verschiedene Lösungsansätze für die Implementierung eines Kamerasystems eingegangen, bevor die Umsetzung detaillierter dargestellt wird. Abschließend wird das Ergebnis betrachtet.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

1.1 Warum Kamerasysteme für Sidescroller-Spiele?

"A great idea can solve multiple problems at the same time."[1]

- Shigeru Miyamoto, Erfinder von Super Mario -

Seit Super Mario Bros. 1985 für Innovation in der Spieleindustrie stand und das Genre der Sidescroller-Spiele mitdefinierte ist viel Zeit vergangen. Dennoch erfreuen sich solche Spiele auch heute noch großer Beliebtheit. So war das von Moon Studios entwickelte und von Microsoft veröffentliche „Ori and the Blind Forest" eines der Highlights der E3 2014[2] und erreichte einen MetaScore von 88[3]. Doch auch unabhängige Entwickler produzieren erfolgreiche Sidescroller-Spiele. So bekam etwa das
Entwicklerstudio Yacht Club Games, welches für sein Spiel „Shovel Knight" per Crowdfunding lediglich um 75.000 $ bat, eine Summe von insgesamt 311.500 $[4].

Ein zentrales Element in Sidescrollern ist das Verhalten der Kamera, die stets den Protagonisten des Spiels im Blick haben muss. Das Verhalten einer Kamera richtig an ein Spiel anzupassen ist eine schwierige Aufgabe, welche viele Ressourcen benötigt. Ressourcen, die viele unabhängige Entwickler, Studenten oder Hobbyisten nicht haben. Das Anpassen der Kamera an sich kann den Entwicklern nicht abgenommen

---

[1] Super Mario Wiki (Hrsg.) (2015): *Shigeru Miyamoto*, veröff. auf mariowiki.com, http://www.mariowiki.com/Shigeru_Miyamoto#Quotes [Zugriff am 09.12.2015]

[2] Hoss, Brian (2014): *Once Secret, Now Known: 'Ori and the Blind Forest' For the Xbox One Shined Brightly at E3*, Online-Artikel, veröff. auf highdefdigest.com, http://www.highdefdigest.com/news/show/games/ori-and-the-blind-forest/moon-studios/Microsoft/Xbox_One/once-secret-now-known-ori-and-the-blind-forest-for-the-xbox-one-shined-brightly-at-e3/15979 [Zugriff am 09.12.2015]

[3] CBS Interactive Inc. (Hrsg.) (2015): Ori and the Blind Forest, veröff. auf metacri-tic.com, http://www.metacritic.com/game/pc/ori-and-the-blind-forest
[Zugriff am 09.12.2015]

[4] Yacht Club Games (Hrsg.) (2014): *Shovel Knight*, veröff. auf kickstarter.com, https://www.kickstarter.com/projects/yachtclubgames/shovel-knight/updates [Zugriff am 09.12.2015]

werden, da sie das endgültige Spiel zu stark definiert. Daher soll ein Plugin entwickelt werden, welches den Entwicklern Grundfunktionen bietet, auf denen im weiteren Entwicklungsprozess aufgebaut werden kann. Ob und wie so ein System funktionieren kann soll im Rahmen dieser Arbeit untersucht werden.

Aufgabe dabei ist es einen Prototypen zu entwickeln, welcher nach dem Basismodell der Softwareentwicklung nach Brandt-Pook und Kollmeier auf eine vorangegangene Konzeption der Software und deren softwarearchitektonischem Design aufbaut.

1.2 Aufgabenstellung

1.2.1 Konzeption

Während des Prozesses der Softwareentwicklung werden verschiedene Phasen durchlaufen. Noch bevor das Projekt realisiert werden kann, müssen verschiedene Vorüberlegungen getätigt werden.

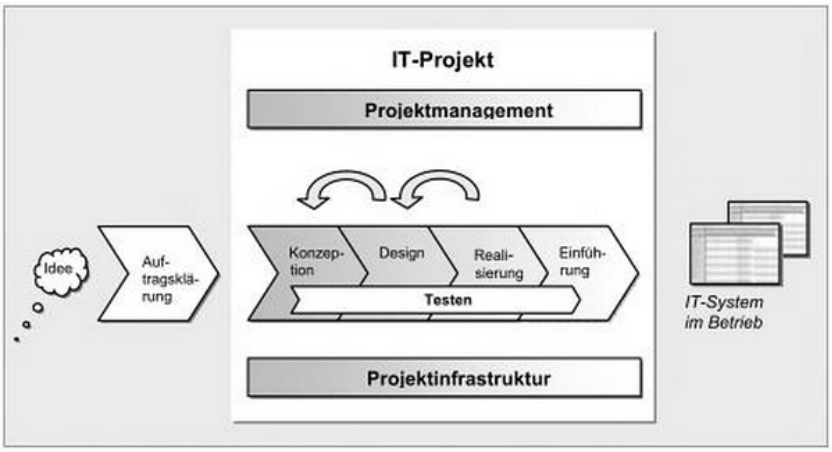


Abbildung 1 - Basismodell der Softwareentwicklung nach Brandt-Pook/Kollmeier[5]

[5] Brandt-Pook/Kollmeier (2008): S. 6

Die Phase der Konzeption klärt die Frage nach dem Leistungsumfang der Software. Der Programmierer muss sich darüber im Klaren sein, welches Ziel oder welche Ziele er mit dem Softwareprojekt erreichen möchte. In der Konzeptionsphase müssen nun die Eigenschaften und Fähigkeiten des Programms erarbeiteten werden, welche für das Erreichen dieses Ziels oder dieser Ziele notwendig sind.

Anschließend bedarf es einer Festlegung der funktionalen Anforderungen, also der notwendigen Funktionen, an die Software. In diesem Rahmen entsteht eine Beschreibung, wie das Projekt die definierten Ziele erreicht.

Des Weiteren ist bei der Konzeption neben der Definierung funktionaler Anforderungen ein Blick auf den Datenhaushalt des Programms vonnöten.

Im Anschluss widmet sich die Konzeptionsphase der Beschreibung und Bewertung grundsätzlicher Lösungsalternativen. Die Nutzwertanalyse, welche Alternativen anhand von Kriterien unterschiedlicher Gewichtung vergleicht, ist dabei die am häufigsten verwendete Methode. Jeder Alternative wird hierbei im Endergebnis ein Nutzwert zugewiesen. Je höher dieser Nutzwert ist, umso wertvoller ist die erdachte Alternative.

Abschließend gilt es in der Phase der Konzeption, ein Lasten- sowie ein Pflichtenheft anzulegen. Während das Lastenheft die Anforderungen aus Sicht des Auftraggebers beschreibt, beschreibt das Pflichtenheft die Leistungen, zu welchen sich der Dienstleister verpflichtet.[6]

[6] Vgl. Brandt-Pook/Kollmeier, 2008: S. 12 ff.

1.2.2 Entwurf

Während die Phase der Konzeption klärt, was die Software leisten soll, widmet sich die Entwurfsphase deren Aufbau. Aufgabe des Designs eines Softwaresystems ist es, die aus den in der Konzeptionsphase erarbeiteten Anforderungen an die Software eine Lösung im Sinne der Softwarearchitektur zu entwickeln[7].

Beim sogenannten High-Level- oder Global-Design wird hierbei zunächst die globale Softwarearchitektur festgelegt. Dies ist der Grobentwurf des Softwaresystems. Im Anschluss werden die einzelnen Subsysteme und Komponenten im Detail entworfen.

Nach Balzert wird dieser Prozess von einer Fülle an sich gegenseitig beeinflussenden Faktoren begleitet. Das Problem beim Entwerfen besteht in der Lösung der durch diese Faktoren entstehenden Probleme, oftmals durch geeignete Kompromisse. Bereits während der Entwurfsphase muss nach Balzert der gesamte von ihm definierte Lebenszyklus des zu entwickelnden Softwaresystems betrachtet werden.

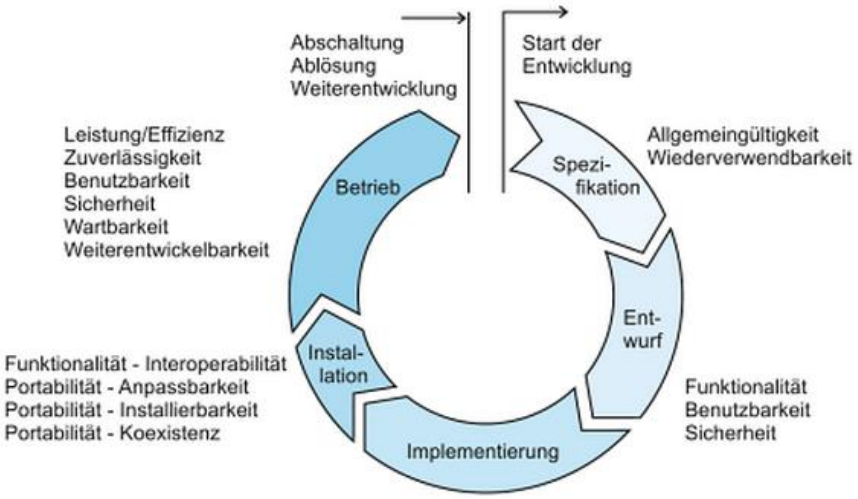


Abbildung 2 - Lebenszyklus eines Softwaresystems nach Balzert[8]

Im strukturierten Entwurf erhalten Module höherer Ebenen allgemeine und steuernde Funktionen, während Module tieferer Ebenen detailliertere Algorithmen enthalten.[9]

---

[7] Vgl. Balzert, 2011: S. 6

[8] Balzert (2011): S. 1

1.2.3 Prototypische Umsetzung

Sind sowohl der Leistungsumfang als auch die Architektur der Software erarbeitet, beginnt die Phase der Realisierung.

Hier wird mit der eigentlichen Programmierung der Software begonnen. In der Realisierungsphase werden Konzeption und Entwurf also in einer Programmiersprache implementiert, daher wird diese Phase auch als Implementierungsphase bezeichnet. Die Realisierung erfolgt in den in der Entwurfsphase erarbeiteten Strukturen. Fallen bei der Realisierung Mängel auf, so muss das Design und gegebenenfalls auch das Konzept überarbeitet werden.[10]

Um bereits frühzeitig eventuelle Mängel erkennen zu können, muss die Software regelmäßig getestet werden. Zum Testen und anschließenden Auswerten der Software empfiehlt sich die Erstellung eines Prototypen.

"Im Rahmen der Informatik, insbesondere in der Softwareerstellung, wird bisher jedes erstmals lauffähige Programm Prototyp genannt."[11]

Dieser Prototyp muss so entwickelt werden, dass alle zuvor definierten Anforderungen an das finale Softwareprojekt erfüllt sind. Im Rahmen des sogenannten Prototyping, also dem Prozess der Prototyp-Erstellung, verfließen die Abgrenzungen der Phasen der Konzeption und des Entwurfs. Der erste lauffähige Prototyp besitzt somit bereits alle wesentlichen Merkmale des finalen Produkts.[12]

[9] Vgl. Schauder/Jarosch/Thieme, 2001: S. 21
[10] Vgl. Brandt-Pook/Kollmeier, 2008: S. 18
[11] Hallmann, 1990: S. 11
[12] Vgl. Hallmann, 1990: S. 23

1.3 Themeneingrenzung

Für den optimalen Untersuchungsablauf bedarf es einer Eingrenzung der in dieser Arbeit zu behandelnden Themen. Mit folgenden zentralen Fragestellungen wird sich dabei im Detail auseinandergesetzt.

- Wie kann ein Pluginn für die Kamerasteuerung in Sidescroller-Spielen aussehen?
- Was muss so ein System können?
- Wie kann ein solches Modell umgesetzt werden?

Kameraführung ist ein weites Feld, selbst wenn es dabei nur um Kameras in Videospielen geht. In der vorliegenden Arbeit wird ganz speziell die Kamera in Sidescrollern behandelt. Dazu wird untersucht, welchen Herausforderungen sich Designer stellen müssen, um eine gute Kamera in ihr Spiel zu integrieren. Dabei wird auch festgestellt, was denn eine „gute Kamera“ ausmacht. Auf Thesen und Erkenntnisse aus der Filmwissenschaft wird dabei nicht eingegangen, da dieser große Bereich die Kapazitäten dieser Arbeit übersteigen würde. Des Weiteren wird diese Arbeit sich nicht dem physikalischen Gebiet der Optik widmen, da Kameras für das Rendern von Spielausschnitten bereits in sämtlichen modernen Game Engines implementiert sind.

Neben der theoretischen Erforschung der Kamera in Sidescroller-Spielen wird in dieser Arbeit dargestellt, wie ein Kamerasystem für solche Spiele konkret aussehen kann. Dafür wird zunächst ein allgemeines Modell geschaffen, anhand dessen ein solches Plugin entwickelt werden kann. Da der zu entwickelnde Prototyp in der Game Engine Unity3D erstellt wird, beziehen sich die detaillierten Darstellungen ausgewählter Aspekte auf diese Engine.

1.4 Methodisches Vorgehen

Nach der Einleitung in Kapitel Eins wird im zweiten Abschnitt der vorliegenden Arbeit zunächst eine Definition des Begriffs Kamera Scrolling erarbeitet. Daraufhin wird die Entwicklung dessen in der Geschichte der Sidescroller-Spiele ergründet, um einen

Überblick über die Bedeutung dieser Technologie zu erhalten. Anschließend werden in Kapitel 2.2 die designtechnischen Herausforderungen beim Scrolling in Videospielen definiert. In diesem Zusammenhang wird in Kapitel 2.3 ein Stufensystem erarbeitet, welches helfen soll, die Herangehensweise verschiedener Spiele an die zuvor definierten Herausforderungen zu erörtern. Dem folgt unter 2.4 ein Überblick über einige gängige Methoden des Kamera Scrollings. Im Kapitel 2.5 werden daraufhin vier Sidescroller-Spiele mit unterschiedlichen Gameplay-Elementen auf ihre Methoden zur Kameraverfolgung hin untersucht. Dabei wird auch auf die zuvor erwähnten Herausforderungen eingegangen. Darauf aufbauend wird in Kapitel 2.6 untersucht, welchen Nutzen ein Kamerasystem für Entwickler haben kann. So kann in 2.7 abschließend herausgefiltert werden, welche genauen Anforderungen an ein zu entwickelndes Kamera-Plugin gestellt werden.

Das dritte Kapitel dieser Arbeit widmet sich der Beleuchtung verschiedener Lösungsansätze für die Entwicklung eines Kamera-Plugins. Im Kapitel 3.1 werden dafür mit dem Lerping und dem Kamerafenster zwei grundlegende Methoden für die Kamerasteuerung miteinander verglichen und gegeneinander abgewogen. Anschließend wird unter 3.2 untersucht, wie ein Parallaxeffekt erzielt werden kann. Dabei wird untersucht, ob es zweckmäßiger ist, die Hintergrundebenen zu bewegen oder mehrere Kameras. Abschließend wird in 3.3 nach einer Möglichkeit gesucht, einem Level Designer die Nutzung des Plugins mittels verschiedener Eingabemöglichkeiten soweit wie möglich zu vereinfachen.

Während sich das zweite und dritte Kapitel der theoretischen Forschung widmen, handelt es sich bei dem vierten Kapitel um die Umsetzung im Detail. Hierfür wird zunächst auf den Kontext des Tools eingegangen. Im Kapitel 4.2 wird in der Struktursicht das Zusammenspiel der verschiedenen Komponenten dargestellt, deren Verhalten unter 4.3 genauer erörtert wird. Unter 4.4 werden zunächst übergreifende Aspekte der Entwicklung beschrieben, bevor die Implementierung des Kameraverfolgungssystems, der regierenden Elemente und des Parallexeffekts genauer beleuchtet werden.

Im abschließenden fünften Kapitel der vorliegenden Arbeit wird zunächst eine umfangreiche Zusammenfassung der erarbeiteten Ergebnisse dargelegt. Anschließend wird der entwickelte Prototyp dargestellt und es wird überprüft, ob dieser sämtliche gestellten Anforderungen erfüllt. Abschließend werden Hinweise für die weitere Forschung auf diesem Gebiet sowie die Erweiterung des aktuellen Prototypen gegeben.

2 Grundlagen

2.1 Kamera Scrolling

2.1.1 Definition

Das englische Wort Scrolling bedeutet zu Deutsch Bildlauf oder Rollen, wird im deutschen Sprachgebrauch gerade im Kontext von Computer- und Videospielen jedoch gemeinhin ebenfalls mit Scrolling übersetzt.

Scrolling ist die - horizontale oder vertikale - gleitende Bewegung von Bild, Video oder Text über einen Bildschirm. Diese Funktion wird genutzt, wenn der anzuzeigende Inhalt zu groß ist, um komplett auf dem entsprechenden Bildschirm dargestellt werden zu können.[13]

So werden beispielsweise längere Dokumente oder Webseiten auf Computern mithilfe von Scrollbars erschlossen, auf mobilen Endgeräte wie Smartphones oder Tablets geschieht dies meist per Fingerwisch.

In Videospielen erlaubt die Nutzung von Scrolling dem Level, größer zu sein als das Spielfenster oder der Bildschirm anzeigen können. Vereinfacht ausgedrückt ist das Scrolling hier die Änderung des sichtbaren Teils der Szenerie des Spiels. Die Kamera fungiert dabei als unsichtbarer, nicht-physischer Akteur, welcher im Spiel wie jeder andere Akteur eine Position besitzt und bewegt werden kann.[14]

[13] Vgl. Janalta Interactive Inc. (Hrsg.) (2015): *Scrolling*, veröff. auf techopedia.org, https://www.techopedia.com/definition/5469/scrolling [Zugriff am 09.12.2015]
[14] Vgl. Stencyl, LLC (Hrsg.) (2015): *The Camera*, veröff. auf stencyl.com, http://www.stencyl.com/help/view/the-camera/ [Zugriff am 09.12.2015]

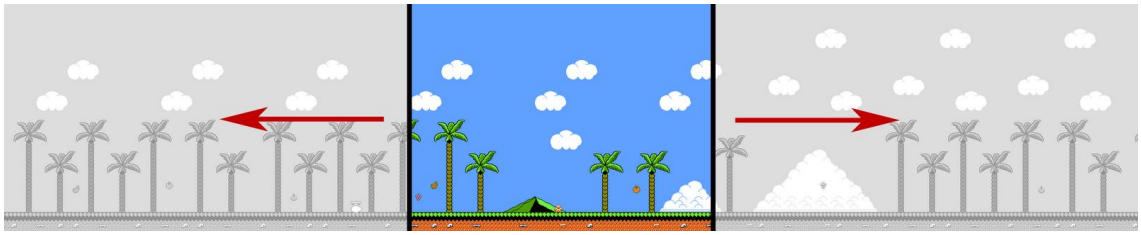


Abbildung 3 - Beispiel für horizontales Scrolling in einem Videospiel[15]

Der Spieler kann durch eine bewegliche Kamera also große Szenerien erschließen. So entstand das Genre des Sidescrollers. Hierbei handelt es sich um Spiele, welche in 2-D oder 2.5-D durch eine Kamera in Seitenansicht betrachtet werden. Als typisches Subgenre des Side Scrollers sind Platformer wie Jump'n'Runs, Beat'em Ups oder auch Shooter zu nennen, in welchen der Spieler rennt, springt, klettert oder schießt. Besonders in den Achtzigerjahren während der Blütezeit dieser Platformer beliebt, werden auch heute noch Spiele mit Side Scrolling-Elementen oder direkt Side Scroller entwickelt.[16]

Für den Einsatz von Scrolling in Videospielen existiert eine Fülle an Möglichkeiten. Die Kamera kann dem Spieler als Fixpunkt in alle Richtungen folgen, um ihn im Mittelpunkt des Geschehens zu arretieren. Das Spiel kann dynamischer gestaltet werden, indem der Hintergrund durch parallaxes Scrolling animiert wird oder einzelne Spielelemente durch eine verfolgende Kamera besonders in Szene gesetzt werden.

Mittlerweile eine über 30 Jahre alte Technik, ist Scrolling noch heute essenziell in vielen Computer- und Videospielen und wird als eine der Basismethoden der Navigation verstanden.

---

[15] Stencyl, LLC (Hrsg.) (2015): *The Camera*, veröff. auf stencyl.com, http://static.stencyl.com/pedia2/ch4/camera/image04.png [Zugriff am 09.12.2015]

[16] Vgl. Janalta Interactive Inc. (Hrsg.) (2015): *Side Scroller*, veröff. auf techopedia.org, https://www.techopedia.com/definition/27153/side-scroller [Zugriff am 09.12.2015]

2.1.2 Einführung in die Anfänge der Sidescroller

Das im Jahr 1972 veröffentlichte Pong ist eines der ersten Arcade-Spiele, welches sich großer Beliebtheit im Mainstream erfreute. Das Spielgeschehen in Pong, in welchem 2 Schläger einen Ball hin- und herschlagen, findet innerhalb der Bildschirmbegrenzungen statt. Das Spielprinzip verlangte hier wie auch in anderen Spielen jener Zeit keinen größeren Bildschirmausschnitt zur Darstellung des Spielinhalts.

Im Jahr 1978 wurde in der Football-Simulation Atari Football erstmals der für den Spieler sichtbare Bereich des Spiels horizontal nach links oder rechts verschoben, um das gesamte Spielfeld eines Footballstadions sichtbar machen zu können.[17]

Damit war der Grundstein für die goldene Äre der Sidescroller in den beiden darauffolgenden Dekaden gelegt.

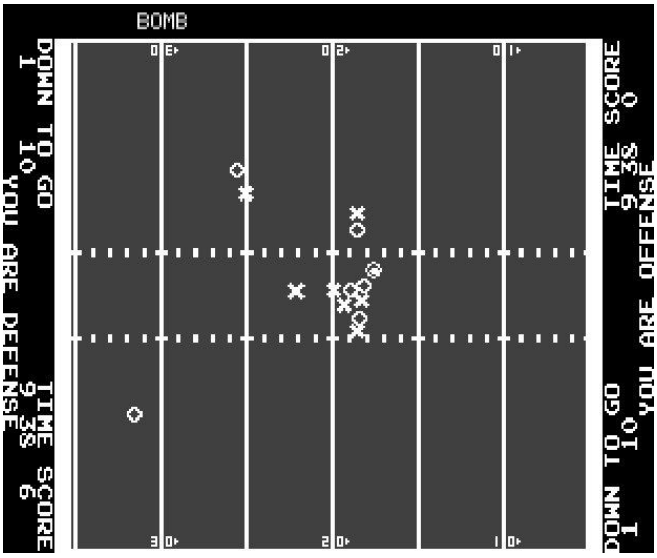


Abbildung 4 - Atari Football (1978)[18]

Die Spiele Super Bug von 1977 sowie Fire Truck von 1978 aus dem Hause Atari verwendeten bereits multidirektionales Scrolling. Hierbei handelte es sich allerdings um

[17] Future US, Inc. (Hrsg.) (2015): *Gaming's most important evolutions. Scrolling*, veröff. auf gamesradar.com, http://www.gamesradar.com/gamings-most-important-evolutions/?page=2 [Zugriff am 09.12.2015]

[18] Future US, Inc. (Hrsg.) (2015): Gaming's most important evolutions. Scrolling, veröff. auf gamesradar.com,http://static.gamesradar.com/images/mb/GamesRadar/us/Features/2010/10/Greatest%20evolutions/atarifootball--article_image.jpg [Zugriff am 09.12.2015]

Racing-Spiele mit Top-Down-Sicht, nicht um klassische Sidescroller. Ähnlich verhält es sich mit dem im November 1980 erschienenen Rally-X von Namco.

Die Shooter Scramble und Defender, beide veröffentlicht im Februar 1981, waren die ersten Sidescroller-Shooter. Der Spieler flog hier jeweils mit einem Raumschiff über eine Planetenoberfläche und musste Aliens abschießen. Defender war so erfolgreich, dass es über 1,2 Milliarden Dollar einspielte.[19]

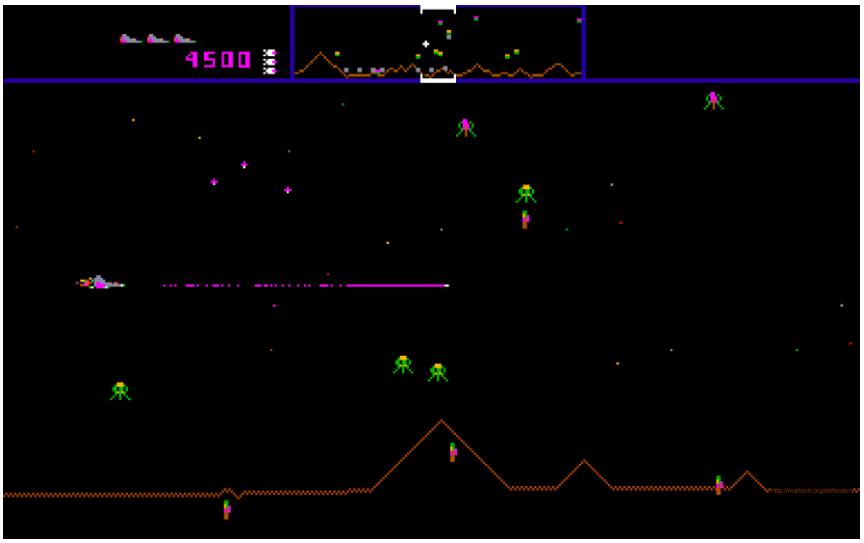


Abbildung 5 - Defender (1981)[20]

Ebenfalls 1981 erschien Jump Bug, welches bereits sanftes horizontales und vertikales Scrolling verwendete. Das Spiel baute sichtbar auf Spiele wie Scramble auf. Um sich von der Konkurrenz zu unterscheiden ließ man den Protagonisten, einen Käfer, im Spielverlauf auf verschiedene Plattformen hüpfen. Der erste Sidescroller-Platformer war geboren.[21]

[19] Vgl. Bogdan, 2014: S. 126

[20] CBS Interactive Inc. (Hrsg.) (2015): *#12 Defender - Williams*, veröff. auf giantbomb.com, http://static.giantbomb.com/uploads/original/24/245897/2678204-3387245354-Defen.png [Zugriff am 09.12.2015]

[21] Ziff Davis, LLC (Hrsg.) (2008): *The Leif Ericson Awards. The Long Jump*, Online-Artikel, veröff. auf ign.com, http://www.ign.com/articles/2008/03/24/the-leif-ericson-awards?page=2 [Zugriff am XX.XX.2015]

Abbildung 6 - Jump Bug (1981)[22]

1982 wurden mit Jungle Hunt und Moon Patrol die ersten Sidescroller veröffentlicht, welche das Parallax Scrolling nutzten. Hier wurden also erstmals unterschiedlich schnell scrollende Layer im Hintergrund übereinandergelegt, um die Illusion von Tiefe und Geschwindigkeit zu erzeugen.[23]

In Moon Patrol übernahm der Spieler einen Buggy, der über die Mondoberfläche fuhr und verschiedenen Hindernissen ausweichen musste.

[22] Ziff Davis, LLC (Hrsg.) (2008): *The Leif Ericson Awards. The Long Jump*, veröff. auf ign.com, http://retromedia.ign.com/retro/image/article/861/861550/the-leif-ericson-awards-20080324022351644.jpg [Zugriff am 09.12.2015]

[23] Future US, Inc. (Hrsg.) (2015): Gaming's most important evolutions. Parallax scrolling, veröff. auf gamesradar.com, http://www.gamesradar.com/gamings-most-important-evolutions/?page=3 [Zugriff am 09.12.2015]

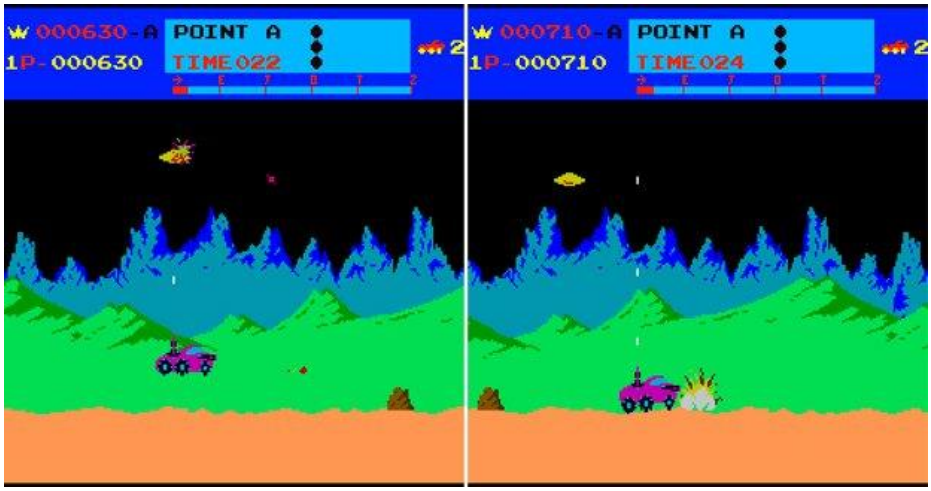


Abbildung 7 - Moon Patrol (1982)[24]

Der 1984 veröffentlichte Sidescroller Hover Attack war frei in alle Richtungen scrollbar. Dies erlaubte es dem Spieler in diesem Shooter, sowohl diagonal als auch geradeaus zu schießen.[25]

Neben Platformern und Shootern beeinflusste das Sidescrolling auch das Genre des Beat 'em Ups. Während es sich beim 1979 erschienenen Warrior noch um ein Vektorspiel mit Draufsicht handelte, erlaubte das horizontale Scrolling die Darstellung größerer Kampfarenen, was gerade für den genretypischen Mehrspielermodus einen großen Fortschritt darstellte. So legte Kung-Fu Master den Grundstein für spätere Beat 'em Ups wie Street Fighter (1987) oder Tekken (1988).[26]

Im August 1984 erschien zudem der Platformer Pac-Land, welcher in seiner Art des Gameplays als wegweisend für spätere Sidescroller-Klassiker wie Super Mario Bros. (1985) oder Sonic the Hedgehog (1991) betrachtet werden kann. Hier finden auch die in dieser Arbeit untersuchten Platformer ihren Ursprung.

---

[24] Future US, Inc. (Hrsg.) (2015): Gaming's most important evolutions. Parallax scrolling, veröff. auf gamesradar.com, http://static.gamesradar.com/images/mb/GamesRadar/us/Features/2010/10/Greatest%20evolutions/moon patrol--article_image.jpg [Zugriff am 09.12.2015]

[25] Ziff Davis, LLC (Hrsg.) (2008): The Leif Ericson Awards. The Long Jump, Online-Artikel, veröff. auf ign.com, http://www.ign.com/articles/2008/03/24/the-leif-ericson-awards?page=2 [Zugriff am XX.XX.2015]

[26] Gamer Network (Hrsg.) (2008): The Tao of Beat-'em'-ups. Part 1: The evolution of a genre, 1976 to 1985, Online-Artikel, veröff. auf eurogamer.net, http://www.eurogamer.net/articles/the-tao-of-beat-em-ups-article?page=2 [Zugriff am 09.12.2015]

2.2 Designtechnische Herausforderungen beim Scrolling

2.2.1 Spielerkontrolle

Die Spielerkontrolle[27] beschreibt, wie stark der Spieler den Kameraausschnitt selbst beeinflussen kann. Dies kann sowohl indirekt als auch direkt geschehen. Eine indirekte Kontrolle wäre zum Beispiel gegeben, wenn die Kamera sich mit dem Charakter mitbewegt, welcher gesteuert wird. Eine direkte Kontrolle ist immer dann gegeben, wenn der Spieler die Kamera mit einer separaten Steuerung unabhängig von dem eigentlichen Spielercharakter bewegen kann. Ein Beispiel dafür lässt sich in „Spelunky“[28] finden. Hier kann der Spieler durch das Drücken einer Pfeiltaste den Bildschirmausschnitt nach unten verlagern.

Ein hoher Grad an Kontrolle durch den Spieler gewährleistet, dass dieser sehen kann, was er sehen möchte. Dies kann jedoch bedeuten, dass er Dinge übersieht, welche der Designer ihn sehen lassen wollte.

2.2.2 Designerkontrolle

Der Designer hat beim Bauen von Welten immer im Sinn, das Auge des Spielers auf bestimmte Punkte zu lenken[29]. Neben der Möglichkeit, diese visuell hervorzuheben, kann er außerdem die Kamera beeinflussen. So kann sich beispielsweise die Kamera beim Erscheinen eines Boss-Gegners auf diesen fokussieren.

Da der Designer nicht live in das Geschehen eingreifen kann, müssen sämtliche gewünschten Effekte und Veränderungen des Kameraverhaltens in das Spiel implementiert werden.

---

[27] Vgl. Keren (2015)

[28] Yu, Derek (2008): Spelunky v1.1 (and Source), veröff. auf tigsource.com, https://forums.tigsource.com/index.php?topic=4017 [Zugriff am 09.12.2015]

[29] Vgl. Keren (2015)

2.2.3 Komfort

Der Komfort beschreibt, wie leicht oder schwer es der menschlichen Physis fällt, sich auf verschiedene Kameraverhalten einzustellen. Um dieses Problem zu erklären wird zunächst der biologische Hintergrund erläutert.

Das Zentrum des gelben Fleckes im menschlichen Auge (Fovea centralis) ist für die detaillierte und scharfe Darstellung zentraler Objekte verantwortlich. Die Gebiete des gelben Flecks hingegen, welche weiter vom Zentrum entfernt sind (Parafovea und Perifovea), reduzieren Bilder und Bewegungen zu Mustern, um Veränderungen schneller realisierbar zu machen. Dies beinhaltet zum Beispiel Veränderungen von Geschwindigkeit und Bewegungsrichtung, aber auch das Erkennen von bekannten Warnsignalen.

Das Vestibuläre System, welches im Innenohr liegt, ist für das Behalten von Balance und für den Orientierungssinn verantwortlich. Dadurch, dass sich der menschliche Körper für die Balance auf die Signale des Vestibulären Systems verlässt, anstatt auf visuelle Signale, kann der Mensch sich selbst bei Änderungen seiner Geschwindigkeit oder Lage im Raum visuell auf ein Ziel fokussieren.

Wenn diese beiden biologischen Systeme gegensätzliche Informationen an das Gehirn senden kann es zu Übelkeit und Schwindel kommen. Dies passiert zum Beispiel durch Lesen während einer Autofahrt oder durch Spielen von Videospielen in sitzender Haltung. Dieser Effekt tritt besonders stark in dreidimensionalen Virtual Reality Spielen auf, ist aber auch in 2D Spielen vorhanden.[30]

[30] Bünte, Hermann (2015): *Vestibuläres System,* veröff. auf med-college.de, http://www.med-college.de/ru/wiki/artikel.php?id=1580&lan=1 [Zugriff am 09.12.2015]

2.3 Eingliederung von Kameralogiken

Um die verschiedenen Kameraverhaltensweisen in Sidescrollern analysieren zu können, wurde ein Stufensystem für die drei Herausforderungen entworfen. Dieses beschreibt, wie stark oder schwach auf einen Bereich eingegangen wird. Dabei muss beachtet werden, dass es sich hierbei nicht um ein Wertungssystem handelt.

Spielerkontrolle

Um zu analysieren wie viel Kontrolle der Spieler über die Kamera hat, wird diese mit der Kontrolle über den Spielercharakter verglichen.

1. Stufe – Spieler hat keinerlei Einfluss auf sein Sichtfeld

2. Stufe – Spieler hat weniger Einfluss auf die Kamera als auf seinen Charakter

3. Stufe – Spieler hat auf das Sichtfeld ebenso viel Einfluss wie auf die Steuerung seines Charakters

4. Stufe – Spieler hat mehr Einfluss auf das Sichtfeld als auf seinen Charakter

5. Stufe – Spieler hat vollen Einfluss auf das Sichtfeld

Designerkontrolle

Für die Analyse der Kontrolle des Designers wird überprüft, wie komplex die Kameraführung ist.

1. Stufe – Designer hat keinerlei Einfluss auf das Sichtfeld des Spielers

2. Stufe – Designer hat nur indirekten Einfluss auf das Sichtfeld des Spielers mittels einer einzigen, allgemeingültigen Logik

3. Stufe – Designer hat nur indirekten Einfluss auf das Sichtfeld des Spielers mittels einer Summe allgemeingültiger Logiken

4. Stufe – Designer hat großen Einfluss auf das Sichtfeld des Spielers mittels individueller Regeln

5. Stufe – Designer hat die komplette Kontrolle über das Sichtfeld des Spielers

Komfort für die menschliche Physis

Der Komfort der menschlichen Physis lässt sich nur schwer einstufen, da jeder Mensch unterschiedlich stark auf die Kameraeffekte reagiert. Da wie in 2.3.3 erläutert Unterschiede zwischen Bewegungen des Sichtfeldes und der Bewegung des menschlichen Körpers, insbesondere Änderungen dieser, Unbehagen hervorrufen, wird hier ausgewertet, wie ruckhafte und unerwartete Änderungen des Kameraverhaltens abgefangen werden. Dabei sind sämtliche Kamerabewegungen als relativ zum Hintergrund zu betrachten, da dieser den größten Teil des Bildschirmausschnitts ausmacht.

1. Stufe – es gibt ausschließlich ruckhafte und/oder unerwartete Kamerabewegungen bei Änderungen des Spielerverhaltens

2. Stufe – es gibt teilweise ruckhafte und/oder unerwartete Kamerabewegungen bei Änderungen des Spielerverhaltens

3. Stufe – ruckhafte und/oder unerwartete Änderungen des Spielverhaltens werden großteils von der Kamera abgefangen

4. Stufe – ruckhafte und/oder unerwartete Änderungen des Spielverhaltens werden vollständig von der Kamera abgefangen

5. Stufe – es existieren keinerlei Änderungen des Kameraverhaltens

2.4 Übersicht gängiger Kameralogiken

Dieses Kapitel zeigt einige gängige Kameralogiken auf. Zusätzlich zu einer knappen Erklärung der Verhaltensweisen[31] wurde das oben entwickelte Stufensystem auf sämtliche Logiken angewendet. Die Anforderungen in der nachfolgenden Tabelle sind die Punkte aus 2.2 dieser Arbeit. Dies soll eine Übersicht über verschiedene Möglichkeiten der Kamerabewegung und deren Effekt geben.

[31] Vgl. Keren (2015)

Tabelle 1 - Kameraverhaltensweisen mit Einstufung

		Anforderungen		
Kameralogik	**Erklärung**	**1.**	**2.**	**3.**
Dem Geschehen folgend				
position-locking	Kamera ist genau auf Spieler gelockt	3	2	1
edge-snapping	Kamera snappt an bestimmter Stelle an eine Wand oder Decke	2	2	2
eingeschränkte Kamerabewegungen				
camera-window	Kamera bewegt sich nur wenn der Spieler sich aus dem Kamerafenster bewegen würde	2	2	3
Snapping				
position-snapping	Kamera driftet mit gleichmäßiger Geschwindigkeit dem Spieler hinterher und versucht an seiner Stelle zu bleiben	2	2	3
platform-snapping	Kamera fokussiert sich in der Höhe nur am Spieler wenn dieser auf einer Plattform landet (Super Mario World)	2	3	3
Smoothing				
zone-smoothing	In festgelegten Kamerazonen folgt die Kamera immer schneller nach	3	2	4
lerp-smoothing	mittels linearer Interpolation folgt die Kamera der Position des Spielers	3	2	4

physics-smoothing	Kamera hat physikalische Eigenschaften wie Trägheit der Maße	2	3	4
Framing				
auto-scrolling	Spieler hat keinerlei Kontrolle über die Kamera, nur über die Position des Charakters innerhalb des Frames	1	5	5
dual forward-focus	mehr Platz in Bewegungsrichtungen	2	2	1
static forward-focus	mehr Platz in genereller Bewegungsrichtung	2	2	1
target-focus	Kamera bewegt sich je nach Geschwindigkeit vom Spieler weg, in Bewegungsrichtung (überholen)	2	2	2
projected-focus	Kamera fokussiert sich auf die Richtung, in welcher ein Zeigeelement vom Spieler entfernt ist	4	1	2
Ausrichtungsänderungen				
camera-path	vorgefertigter Pfad der Kamera durch das Level	1	5	5
region-based	Änderung der Kamera je nach Region	1	4	n.a.
zoom-to-fit	Kamera versucht so nah wie möglich an allen relevanten Features zu sein	2	3	2
cue-focus	Fokus wird von Levelelementen beeinflusst	2	4	3
gesture focus	Game Play triggert bestimmte Kamerabewegungen	2	4	3
cinematic breaks	Kamera bricht für cineastische Szenen aus normalem Verhalten aus	1	5	n.a.

Multifokale Kamera				
position averaging	Fokus auf der durchschnittlichen Position aller Kamera-Ziele	2	2	2

manual control	Spieler hat selbst Kontrolle über die Kamera - nicht zwangsläufig vollständig	4	2	2
camera shake	Kamera wackelt unkontrolliert	1	2	1

2.5 Analyse der Kamerasysteme aktueller Spiele

Im Folgenden werden vier aktuelle Sidescroller-Spiele auf ihr Kameraverhalten hin analysiert. Dazu wird zuerst ermittelt, welche Kameraerhaltensweisen in den Spielen verwendet werden. Dazu wurden während des Spielens Videos aufgenommen. Diese Videos wurden anschließend geschnitten, um Cut Scenes und Ladebildschirme aus dem Material zu entfernen. Anschließend wurde mit einer Videobearbeitungs-Software ein Kreuz an den Mittelpunkt des Bildausschnittes gesetzt. Dadurch lässt sich einfacher beobachten, auf welchen Punkt des Bildes die Kamera fokussiert ist, da das Auge somit einen Fixpunkt hat, an welchem es sich orientieren kann. Somit fällt die Beobachtung der Kamerabewegungen leichter. Die Zeitangaben in den Analysen beziehen sich dabei auf das jeweils dazugehörige Video, welches im digitalen Teil der Arbeit zu finden ist.

Nachdem die individuellen Verhaltensweisen der Kamera ermittelt sind, wird das in 2.3 entwickelte Stufensystem darauf angewendet. Schlussendlich wird der Effekt der Kameralogiken den Anforderungen des Gameplays an das Kameraverhalten gegenübergestellt.

Die analysierten Spiele wurden ausgewählt, da sie ein sehr unterschiedliches Gameplay und somit sehr unterschiedliche Anforderungen an das Kameraverhalten haben.

2.5.1 Mutant Mudds Deluxe

Das Spiel „Mutant Mudds Deluxe“[32] erinnert stark an frühe Spiele aus der „Mega Man“- Serie[33]. Es handelt sich um ein Plattformer-Spiel in Pixel-Optik, welches einen gewissen Retro-Effekt[34] beim Spieler auslöst. Die hauptsächlichen Gameplay-Elemente sind das Springen von Plattform zu Plattform und das Beseitigen von Gegnern mittels einer abstrakten Feuerwaffe. Neben diesen bekannten Mechaniken gibt es noch eine Innovation. So ist es dem Spieler möglich, an bestimmten Stellen der Level in den Vorder- oder Hintergrund zu springen und sich in der jeweiligen Ebene weiterzubewegen. So werden die aus früheren Spielen bekannten parallaxen Zierebenen bespielbar. Generell handelt es sich hier um ein recht schweres Spiel. Die Sprünge in späteren Levels verlangen pixelgenaues Springen, was durch Gegner und „Hazards“[35] noch erschwert wird. Der Spieler muss ein gutes Zeitgefühl mitbringen um die Level zu meistern.

Die Kamera des Spiels fokussiert sich fast ausschließlich auf den Spieler. Dies ist besonders gut an den Sprüngen zwischen 1:22 und 1:26 zu erkennen. Der Fokus der Kamera bleibt dabei genau auf dem Spieler, sowohl in der Horizontalen als auch der Vertikalen. Gerade der Anfang des Videos lässt allerdings anderes vermuten. Zwischen 0:00 und 0:13 liegt der Fokuspunkt der Kamera rechts vom Spieler und folgt diesem erst wenn er den Fokuspunkt erreicht hat. Auch auf der Höhenachse folgt die Kamera dem Spieler erst, wenn seine Sprunghöhe über den Fokuspunkt hinausgeht. Ein ähnliches Verhalten lässt sich zwischen 1:03 und 1:06 beobachten. Hier folgt die Kamera dem Spieler in der Höhe ab einem gewissen Punkt nicht mehr, sondern erst wieder, wenn er auf die Höhe des Fokuspunktes herabgesunken ist. Dies lässt sich sehr einfach erklären. Zu Beginn des Spiels steht der Spieler an der linksseitigen Begrenzung und am tiefsten Punkt des Levels. Zwischen 1:03 und 1:06 ist der Spieler am oberen Rand des Levels. Die Kamera snappt also an den Levelbegrenzungen und verhindert somit, dass unwichtige Dinge gezeigt werden. Zu Beginn des Levels signalisiert sie dem Spieler somit außerdem, dass er sich nicht nach Links bewegen kann. Besonders deutlich wird das Snappen an den Levelrändern zwischen 1:49 und 2:03. Hier bewegt sich der Spieler deutlich oberhalb des Bildschirmmittelpunktes.

---

[32] Renegade Kid LLC (Hrsg.) (2012): *Mutant Mudds Deluxe*, veröff. auf renegadekid.com, http://www.renegadekid.com/mutantmudds.htm [Zugriff am 09.12.2015]

[33] Capcom AG (Hrsg.): *Mega Man Zero Official Complete Works,* 2008

[34] Vgl. Fenty (2008), S. 23 und Felzmann (2012), S. 70

[35] Vgl. Rogers (2010), S. 126

Ein Blick in die Tabelle aus 2.4 zeigt, dass die verwendeten Kameratechniken des Position-Locking und des Edge-Snapping zu den einfachsten Methoden der Kameraverfolgung zählen. Die Kontrolle des Spielers über die Kamera ist genauso groß wie seine Kontrolle über die Spielfigur, mit Ausnahme der Levelbegrenzungen. Hier hat er etwas weniger Kontrolle über die Kamera. Die Kontrolle des Spielers schwankt also zwischen der zweiten und dritten Stufe.

Die einzige Kontrolle, welche der Designer über den Fokus der Kamera hat, ist indirekter Natur. Auf der einen Seite kann er beim Level Design bestimmen, wohin sich der Spieler bewegen kann, womit gleichzeitig festgelegt wird, wohin die Kamera sich bewegen kann. Außerdem bestimmt er die Begrenzungen des Levels und beeinflusst somit, an welchen Stellen die Kamera an den Rändern snappt. Er hat somit nur mittels einer einzigen definierten Kameralogik, dem Position-Locking mit Edge-Snapping[36], indirekten Einfluß auf das Kameraverhalten. Dies entspricht der zweiten Stufe in der Designerkontrolle.

Der Komfort des Spielers rückt bei den ermittelten Kameraverhaltensweisen in den Hintergrund. Mit Ausnahme der Levelränder arretiert die Kamera auf dem Spieler, wodurch es zu unsanften Bewegungen der Kamera kommt. Jede Bewegung des Spielers wird sofort auf die Kamera übertragen. Somit ist ein Spielerkomfort der Stufe Eins gegeben, an den Levelrändern der Stufe Zwei. Dies hat allerdings keine allzu negativen Auswirkungen, da das gesamte Spielgeschehen recht langsam abläuft.

Die Einfachheit der Kameralogik geht einher mit dem Retrogefühl, welches das Spiel sowohl optisch, als auch vom Gameplay her vermitteln soll. Das Arretieren der Kamera auf den Spieler ist nicht auf Komfort ausgelegt, vereinfacht dem Spieler aber die herausfordernden Sprungpassagen. Durch die gewählte Kameraführung wird der Charakter des Spiels unterstützt.

Gesondert erwähnt werden muss noch das Level Design mit den Parallax-Layern. Hier muss der Designer darauf achten, dass Vorder- und Hintergrund genau auf den Mittelgrund des Spiels abgestimmt sind. Dies erschwert das Design, ermöglicht es dem Designer aber auch, Geheimnisse einzubauen, welche der Spieler nur einsammeln kann, wenn er sie von einer anderen Ebene aus entdeckt.

[36] Das Edge-Snapping kann nicht als alleinstehende Methode verwendet werden, dennoch wird es in dieser Arbeit immer separat aufgeführt, da es das Verhalten der Kamera ändert.

2.5.2 Ori and the Blind Forest

„Ori and the Blind Forest“[37] ist ein Spiel, welches sich aufgrund seiner komplexen und verzweigten Level und der Spielmechanik mit verschiedenen Skills zur Kategorie der Metroidvanias[38] zuordnen lässt. Es ist für Hardcore-Spieler gedacht, welche an sehr anspruchsvollen Spielen wie eben „Metroid“ und „Castlevania“ Freude haben. Wie auch in diesen Spielen liegt der Fokus des Spiels auf schwierigen Kämpfen und der Herausforderung, sehr komplexe Level zu erkunden. Dabei wird ein kleiner, niedlicher Waldgeist gespielt, welcher in einer düsteren und bedrohlich wirkenden Welt vorankommen muss. Der Waldgeist wurde von den Designern sehr flink und dynamisch gestaltet. Dies spiegelt sich neben dem Character Design, den Animationen und seinen Fähigkeiten auch in einer hohen Bewegungsgeschwindigkeit wider.

Die Kamera folgt dem Spieler in diesem Spiel nicht statisch. Vielmehr passt sie ihre Verfolgung zu jedem Zeitpunkt dynamisch an. Dies bezieht sich sowohl auf den Verfolgungspfad als auch die Geschwindigkeit. Die Geschwindigkeit der Kamera wird aus der Entfernung dieser zum Spieler ermittelt. Dabei ist die Kamera umso schneller, je größer ihre Entfernung zum Spieler ist. Dadurch, dass die Kamera bei geringer Entfernung langsamer als der Spieler ist, fällt sie zunächst hinter ihm zurück. Ab einer gewissen Entfernung hat die Kamera jedoch die gleiche Geschwindigkeit wie der Spieler erreicht und kann bei normaler Bewegung somit nicht noch weiter zurückfallen. Bleibt der Spieler nun stehen, holt die Kamera ihn wieder ein, wobei sie umso langsamer wird, je näher sie an ihn herankommt. Auf diese Technik wird im Kapitel 3.1.2 näher eingegangen.

Der Pfad, auf dem sich die Kamera hinter dem Spieler her bewegt, wird zu jedem Frame neu berechnet und ist jeweils der gerade Pfad zwischen den aktuellen Koordinaten der Kamera und denen des Spielers.

Neben dieser Technik wird auch das bereits aus 2.5.1 bekannte Edge-Snapping verwendet, was zwischen 0:05 und 0:42 erkennbar ist.

---

[37] Moon Studios (Hrsg.) (2015): *Ori and the Blind Forest*, veröff. auf oriandtheblindforest.com, http://www.oriblindforest.com/#!// [Zugriff am 09.12.2015]

[38] Parish, Jeremy (2009): *Metroidvania: Rekindling a Love Affair with the Old and the New*, veröff. auf 1up.com, http://www.1up.com/do/blogEntry?bId=8999257&publicUserId=5379721 [Zugriff am 09.12.2015]

Um die Auswirkungen der Kamera einzuschätzen hilft wieder ein Blick in die Tabelle aus 2.4, in welcher die beschriebene Technik der Kameraverfolgung als Lerp-Smoothing aufgeführt ist. Da die Kamera dem Spieler, wenn auch mit Verzögerung, überall hin folgt, hat er genau so viel Kontrolle über diese, wie über den Spielercharakter, was der Stufe Drei der Spielerkontrolle entspricht. Da der Designer in diesem Spiel nicht mehr und nicht weniger Kontrolle über die Kamera als im Spiel „Mutant Mudds“ hat, ist dieses Beispiel ebenfalls der zweiten Stufe der Designerkontrolle zuzuordnen. Dahingegen unterscheidet sich der Spielerkomfort drastisch im Vergleich zum vorherigen Beispiel. Durch die sanfte Kameraführung entstehen keinerlei ruckhafte Bewegungen, was der Stufe Vier des Spielerkomforts entspricht.

Die dynamischen und sanften Kamerafahrten unterstreichen den Charakter des Protagonisten, wodurch sich das Spiel „Ori and the Blind Forest“ sehr flüssig spielt. Der Spieler hat auch bei kurzen Pausen, in denen er seine Spielfigur nicht bewegt, das Gefühl, es würde etwas passieren, da die Kamera noch in Bewegung ist. Zwar ist das Spiel sehr stark auf das Entdecken der Level und ihrer Geheimnisse ausgelegt, dennoch bekommt der Spieler durch die Kamera keine Hinweise auf deren Verbleib. Hierdurch wird an die Historie der Metroidvania-Spiele angeschlossen, welche dem Spieler auch keinerlei Hilfestellung gaben. Dies ist ein entscheidender Unterschied zum nächsten Beispiel.

2.5.3 The Cave

In „The Cave“[39] muss der Spieler Rätsel lösen, indem er Level erkundet und darin befindliche Gegenstände findet, miteinander kombiniert und einsetzt. Dies ist eigentlich Gameplay, welches aus klassischen Point&Click-Adventures[40] bekannt ist. Da der leitende Entwickler von „The Cave“ Ron Gilbert war, welcher für „Monkey Island“[41] bekannt geworden ist, ist es nicht weiter verwunderlich, dass diese Mechaniken jetzt in diesem Sidescroller-Spiel vorkommen. Ähnlich wie in seinen bisherigen Spielen zeichnet sich auch dieses durch eine komische und lustig erzählte Geschichte aus. So liegt der Hauptfokus des Spiels ganz klar auf dem Lösen von Rätseln und es gibt kaum Sequenzen im Spiel, welche vom Spieler schnelle Reaktionen oder gutes Zeitgefühl verlangen.

Die Bewegungen der Kamera an sich funktionieren nach dem gleichen System des Lerpings wie in dem vorangehenden Beispiel. Bei genauerer Betrachtung fällt jedoch auf, dass die Kamera sich selbst bei längerem Stillstand nicht immer auf den Protagonisten zentriert. Stattdessen wird der Fokuspunkt der Kamera an vielen Stellen zu anderen Objekten hingezogen, wie zwischen 0:22 und 0:32 zu dem kaputten Schaltkasten in einer tieferen Ebene. Es handelt sich hierbei um einen Cue-Focus, wie er in 2.4 in der Tabelle dargestellt ist. An dieser Stelle zoomt die Kamera sogar hinaus, um dem Spieler dieses Element besser zu zeigen.

Diese Elemente sind völlig individuell in den Leveln gesetzt und haben den Nutzen, dem Spieler Hinweise zum Lösen der Rätsel zu geben. Durch die Individualität hat der Designer eine große Kontrolle über die Position der Kamera, was der Stufe Vier für die Designerkontrolle entspricht. Notwendigerweise wird dem Spieler damit allerdings die zu einem gewissen Grad die Kontrolle entzogen. Entsprechend der Beschreibung von Stufe Zwei der Spielerkontrolle hat er zwar Kontrolle über die Kamera, allerdings zu einem geringeren Maß als über seinen eigenen Spielercharakter. Dadurch können

---

[39] Gilbert, Ron/Double Fine Productions, Inc. (2013): *The Cave*, veröff. auf thecavegame.com, http://thecavegame.com/about.html [Zugriff am 09.12.2015]

[40] Vgl. McConville, Ciaran (2015): *Point-and-Click Adventure Games Are Dead*, veröff. auf evansreview.com, http://evansreview.com/science-technology/point-and-click-adventure-games-are-dead/11677 [Zugriff am 09.12.2015]

[41] The Monkey Island SCUMM Bar (Hrsg.) (2004): *The Secret of Creating Monkey Island - An Interview With Ron Gilbert, excerpt from LucasFilm Adventurer vol. 1, number 1, Fall 1990,* veröff. auf scummbar.com, http://scummbar.com/resources/articles/index.php?newssniffer=readarticle&article=1033 [Zugriff am 09.12.2015]

einige Kamerabewegungen für den Spieler unerwartet kommen. Durch die lineare Interpolation werden diese aber zum Großteil sanft gestaltet, wie in Stufe Drei für Komfort beschrieben.

Zusätzlich zu den normalen Kameraverfolgungen gibt es in diesem Spiel außerdem cineastische Kamerafahrten, welche dem Spieler Hinweise zum Lösen der Rätsel geben. Während dieser hat der Spieler keinerlei Kontrolle, weder über seine Spielfigur, noch über die Kamera.

Beim Lösen der Rätsel, dem Hauptelement des Spiels, wird der Spieler durch die Technik des Cue-Focus unterstützt. Die sanften Kamerafahrten erlauben ein entspanntes Lösen der Rätsel, welches den Spieler nicht physisch belastet. Trotz einer einfachen Technik ist es den Designern gelungen, die komplexen Level und Rätsel für den Spieler etwas einfacher zu machen, wodurch er die Komik der Geschichte leichter genießen kann.

2.5.4 Never Alone

Ähnlich wie im vorherigen Beispiel ist die Erzählung einer Geschichte auch ein zentraler Punkt im Spiel „Never Alone“[42]. Der Spieler kann die beiden Protagonisten, ein kleines Inuitmädchen und einen Polarfuchs, spielen und während des Spiels dynamisch zwischen beiden wechseln, um leichte Rätsel zu lösen. Alternativ kann das Spiel auch zu zweit gespielt werden, wobei jeder Spieler eine Spielfigur übernimmt.

Anders als in den vorherigen Beispielen kommt hier ein deutlich komplexeres Kamerasystem mit mehreren Komponenten zur Anwendung.

Das Spiel hat verschiedene Arten von Abschnitten, welche sich unterschiedlich spielen.

1. Spielsequenzen: In diesen spielt der Spieler komplett eigenständig und die Kamera folgt festen Regeln

2. Cineastische Sequenzen: In diesen wird ein Videoclip abgespielt. Dies wird vom Spieler zunächst nicht wahrgenommen, da er in den Clip hineinläuft und noch selbst die Spielfigur kontrolliert. Dann wird ihm die Kontrolle abgenommen, doch die Figur bewegt sich zunächst weiter wie bisher, der Punkt an dem der Spieler selbst nicht mehr spielt ist somit schwer zu bemerken. Neben der Kontrolle über seine Figur verliert der Spieler außerdem sämtliche Kontrolle über die Kamera.

3. Halb-Cineastische Sequenzen. Hier spielt der Spieler die Figur zwar selbst, aber die Kamera folgt anderen Regeln als während der restlichen Abschnitte. Dies dient dem Aufbau einer gewisse Dramatik[43] aufzubauen.

Um die Kamera zu analysieren müssen diese Sequenzen separat voneinander betrachtet werden.

---

[42] Upper One Games LLC (Hrsg.) (2014): *Never Alone (Kisima Ingitchuna)*, veröff. auf neveralonegame.com, http://neveralonegame.com/game/ [Zugriff am 09.12.2015]

[43] An dieser Stelle ist vom Autor keine wissenschaftliche Definition gemeint, sondern der umgangssprachliche Gebrauch des Wortes

Spielsequenzen

Um das Konzept hinter der Kameraverfolgung zu verstehen bedarf es einer detaillierten Betrachtung des Videoclips zwischen 1:21 und 2:00. Das Video wurde im Einzelspielermodus aufgenommen, in welchem die zweite Spielfigur der ersten automatisch folgt. In diesem Fall das Mädchen dem Fuchs. Wie zu beobachten ist, folgt die Kamera dem Spieler nicht immer gleich schnell. Wenn sich der Fuchs schnell bewegt, folgt auch die Kamera schnell. Bewegt er sich hingegen langsam, folgt auch die Kamera langsam. Bei schrittweiser Fortbewegung folgt die Kamera zunächst sogar gar nicht, sondern erst später. Es lässt sich also konstatieren, dass die Kamera geschwindigkeitsbezogen reagiert. Die Geschwindigkeit des Spielers bestimmt also die Geschwindigkeit der Kamera. Zusätzlich dazu wird die Kamera aber auch von ihrer eigenen Geschwindigkeit beeinflusst. Bewegt sie sich selbst langsam, holt sie den Spieler nicht vollständig ein, sondern erreicht ihn nicht immer. Hat die Kamera eine hohe Geschwindigkeit erreicht, überholt sie den Spieler sogar und verlangsamt sich erst dann. Dieses Verhalten entspricht dem eines Autos am Abschleppseil. Die Kamera ist also eine eigene physische Entität und folgt dem Trägheitsgesetz der Masse.

Diese Möglichkeit, die Kamerabewegungen mit physikalischen Gesetzen sanft zu machen, sorgt dafür, dass ruckhafte und unerwartete Bewegungen komplett abgefangen werden (Stufe 4 Komfort). Allerdings haben weder der Spieler, noch der Designer viel Einfluss auf die Kamera. Die Spielerkontrolle erreicht somit die Stufe Zwei. Da in diesem Spiel, wie auch bei „The Cave“, außerdem Cue-Foki verwendet werden, erlangt der Designer die Kontrolle über die Kamera zurück und erreicht somit die Stufe Vier für Designerkontrolle.

Im Mehrspielermodus wird die gleiche Kameratechnik angewendet, hier wird allerdings keiner der beiden Spieler als Ausgangspunkt für die Kamera genommen, sondern der Mittelpunkt zwischen den Beiden. Sollten sich die beiden Spieler weit voneinander entfernen, zoomt die Kamera heraus, um beide im Bild zu behalten (Videoclip 2:21 bis 2:29).

Cineastische Sequenzen

Eine solche Sequenz existiert zwischen 0:46 und 0:58 des Videoclips. Hier hat der Spieler keinerlei Kontrolle über das Spiel oder die Kamera. Stattdessen sind die Bewegungen dieser komplett vom Designer vorgefertigt. Dies entspricht der Stufe Eins für Spielerkontrolle und der Stufe Fünf für Designerkontrolle. In dem vorliegenden Beispiel ist die Kamera außerdem sehr komfortabel für den Spieler, jedoch kann dies

nicht zwangsweise für sämtliche cineastischen Sequenzen festgelegt werden, da diese vom Designer auch anders hätten gestaltet sein können.

Halbcineastische Sequenzen

In diesen Sequenzen wird zwar die gleiche grundlegende Technik wie in den normalen Spielsequenzen verwendet, allerdings gelten zusätzliche Regeln. So ist zu Beginn des Videoclips die Verfolgung des Mädchens durch den Eisbären zu sehen. Dieser hat eine Gewichtung erhalten, welche die Kamera mit zu ihm hin zieht. Der Fokuspunkt der Kamera wird durch den gewichteten Mittelpunkt des Mädchens und des Eisbären bestimmt, wobei die Wichtung des Mädchens höher ist. Zusätzlich wird der Fokus der Kamera nur auf der Querachse dynamisch bestimmt, in der Höhe folgt sie hingegen einem Pfad, welcher über den beiden handelnden Personen liegt.

Der Designer überlässt sich in diesen Sequenzen mehr Kontrolle über die Kamera, um das Geschehen so gut wie möglich in Szene zu setzen.

Der große Fokus auf der Erzählung und Inszenierung kommt bei der Kamera deutlich zum Vorschein. So wirkt die Kamera sehr natürlich und erleichtert dem Spieler das Verfolgen der Handlung. Da es in „Never Alone“ keinerlei schwierige Passagen gibt, in welchen der Spieler sich auf Timing und Reaktion konzentrieren müsste, gibt es auch keine Punkte, an denen ihn die träge Kamera beim Spielen behindern würde.

2.6 Schlussfolgerungen über den Nutzen eines Kamerasystem

Bei einer Gesamtbetrachtung der Analysen fallen mehrere Dinge auf. Zum einen ist jede Kamera sehr gut an das Spiel angepasst. In jedem Beispiel unterstützt die Kamera die Hauptfokuspunkte des jeweiligen Spiels. Somit liegt der Schluss nahe, dass jedes Sidescroller-Spiel eine eigens angepasste Kamera benötigt. Des Weiteren unterscheiden sich die Kameras der Spiele zum Teil drastisch voneinander. Dies wird besonders deutlich, wenn man die harten Kamerabewegungen in „Mutant Mudds“ mit den physikalisch korrekten Bewegungen in „Never Alone“ vergleicht. Dennoch gibt es auch wiederkehrende Elemente, wie die Verwendung von Cues oder das Edge-Snapping an Levelrändern. Aus den Analysen der Spiele lassen sich folgende

Schlussfolgerungen für die Implementierung eines Kamerasystems treffen.

Designer benötigen die Möglichkeit, verschiedene Kameras testen zu können.

Um eine Kamera vollständig an ein Spiel anpassen zu können, müssen verschiedene Techniken der Kameraverfolgung ausprobierbar sein. Auf diese Weise kann entschieden werden, welche Technik das Spielgefühl am besten unterstützt.

Die Kamera muss individuell einstellbar sein.

Um dem Spieler einen möglichst hohen Komfort und eine passende Mischung aus Spielerkontrolle und Designerkontrolle zu bieten, genügt es nicht, sich nach den unter 2.3 entwickelten Stufen zu richten. Diese können als Richtlinie gelten. Dennoch entscheidet am Ende das Feintuning darüber, ob das Spielgefühl richtig vermittelt wird. Hierzu müssen sinnvolle Parameter entwickelt werden.

Das Kamerasystem muss erweiterbar sein.

Wie festgestellt wurde, unterscheiden sich die Kameras stark voneinander. Betrachtet man die unter 2.1.2 dargestellte Geschichte der Sidescroller-Kameras unter dem Aspekt der ständigen Entwicklung von neuen Techniken, muss ein System so offen gestaltet sein, dass es den Entwicklern, die es nutzen, möglichst ist, immer weitere Techniken in dieses zu integrieren.

2.7 Anforderungen an das Kamerasystem

Nachdem in den vorherigen Kapiteln analysiert wurde, was ein Kamerasystem leisten muss, werden jetzt anhand der ISO9126[44] folgende Anforderungen an das Kamerasystem gestellt:

1. Installierbarkeit: Das Plugin soll von einem Designer ohne Programmiererfahrung in die jeweilige Engine installiert werden können.

2. Anpassbarkeit: Das System selbst soll von einem Programmierer mit Erfahrung in der jeweiligen Game Engine an die Bedürfnisse der Designer angepasst werden können.

3. Koexistenz: Das System soll unabhängig von anderen Systemen innerhalb eines Spiels neben diesen funktionieren, ohne dass Anpassungen vorgenommen werden müssen.

4. Effizienz: Das System soll einem kleinen Entwicklerteam den größten Arbeitsteil der Implementierung einer Kamera in ihr Sidescroller-Spiel abnehmen und Designern möglichst viele Mittel an die Hand geben, um Programmierer zu entlasten.

5. Wiederherstellbarkeit: Bei versehentlicher Überschreibung oder Fehlern durch falschen Gebrauch soll das System ebenso einfach installierbar sein wie bei der ursprünglichen Installation.

6. Verständlichkeit: Das Programm soll für einen Designer, welcher Grundkenntnisse zu Kameratechniken in Sidescrollern hat, gemeinsam mit einem Handbuch zu bedienen sein.

---

[44] Vgl. Johner, Christian (2015): *ISO 9126 und ISO 25010*, veröff. auf johner-institut.de, https://www.johner-institut.de/blog/iec-62304-medizinische-software/iso-9126-und-iso-25010/ [Zugriff am 09.12.2015]

7. Erlernbarkeit: Das Erlernen des Programms in seinen Grundzügen soll nicht mehr als einen Tag in Anspruch nehmen.

8. Bedienbarkeit: Das Programm soll mit nichts weiter als Computermaus und den Zahlen auf einer Tastatur bedienbar sein.

3 Lösungsvarianten für die Umsetzung des Kamerasystems

3.1 Methodik der Kameraverfolgung

Um mit der Kamera der Spielfigur zu folgen gibt es mehrere Möglichkeiten. In den ersten Sidescroller-Spielen führte die Kamera einfach nur die gleiche Bewegung wie der Spieler aus, wodurch dieser immer fokussiert war. Diese ständige Bewegung der Kamera, und damit des Hintergrundes, ist jedoch nicht komfortabel für die menschliche Physis. Um den Komfort zu erhöhen wurden im Laufe der Zeit verschiedene Techniken entwickelt, welche die Kamerabewegungen einschränken. An dieser Stelle sollen zwei grundlegende Techniken betrachtet werden.

3.1.1 Kamerafenster

Das Kamerafenster[45] war einer der ersten Versuche, die Bewegungen der Kamera zu minimieren. Das Prinzip dahinter ist simpel. Solange sich der Spieler innerhalb eines festgelegten Bereiches, eines „Fensters“, bewegt, folgt ihm die Kamera nicht. Erst wenn der Spieler diesen Bereich verlassen würde bewegt sich die Kamera mit ihm mit.

Dies ist sehr einfach zu realisieren. Hierfür muss der Quellcode für die Übertragung der Bewegung des Spielers auf die Kamera nur dann durchgeführt werden, wenn sich die Koordinaten des Spielers einen festgelegten Wert von den Koordinaten der Kamera entfernt befinden. So folgt die Kamera beispielweise dem Spieler in der Höhe, wenn die Differenz der Werte der beiden Höhenkoordinaten von Spieler und Kamera einen vom Designer festgelegten Wert erreicht.

[45] Vgl. Keren (2015)

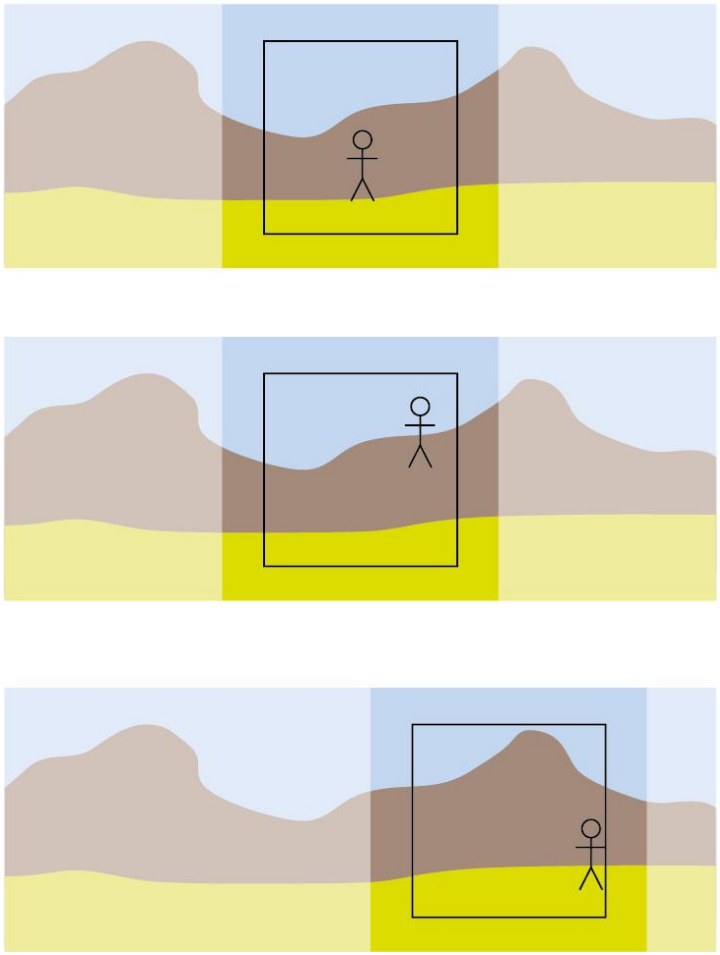


Abbildung 8 - Funktionsweise eines Kamerafensters

Das Ergebnis ist eine Kamera, welche sich so selten wie möglich bewegt. Von Nachteil ist, dass der Spieler bei der Bewegung in eine bestimmten Richtung ein verkleinertes Sichtfeld hat. Hierbei nimmt die Blickweite in Bewegungsrichtung umgekehrt proportional zur Größe des Bereichs ohne Kamerabewegung ab.

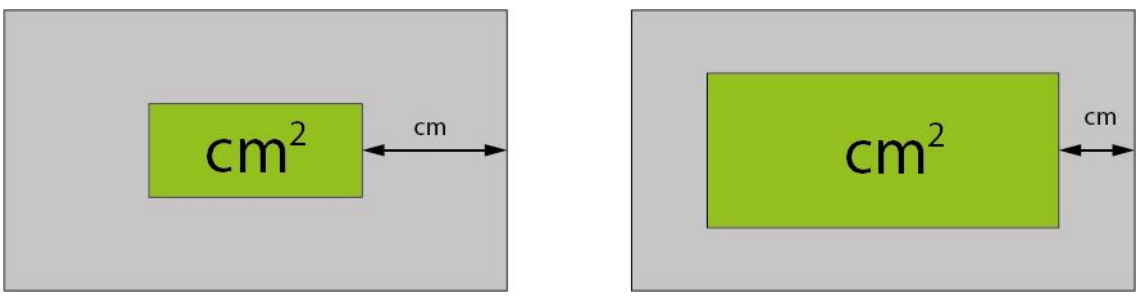


Abbildung 9 - Ausmaße von Kamerafenstern

3.1.2 Lerping

Das Lerping[46] isteine Funktion, welche sich der mathematischen, linearen Interpolation bedient. Bei dieser Funktion wird ein Punkt auf einer Geraden zwischen zwei bekannten Punkten gesucht. Die mathematische Formel für die lineare Interpolation sieht wie folgt aus.

$$y = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1)$$

Hier wird mittels zweier gegebener Punkte der Y-Wert zu einem gegebenen X-Wert gesucht. Die Funktion .lerp in Unity sieht wie folgt aus.

public static Vector3 Lerp(Vector3 a, Vector3 b, float t);

Hier wird ein Punkt zwischen zwei dreidimensionalen Vektoren (Vector3 a und Vector3 b) gesucht, welcher sich zu einem gegebenen Anteil (float t; wobei t zwischen 0 und 1 liegt) vom Startvektor (Vector3 a) entfernt befindet.

Wird diese Funktion nun in einem Skript verwendet, um ein Objekt in vielen kleinen (für den Menschen nicht wahrnehmbaren Schritten) von einem Punkt zu einem anderen zu, bewegen kann dies wie folgt aussehen.

[46] Vgl. Unity Technologies (Hrsg.) (2015): *Vector3.Lerp,* veröff. auf docs.unity3d.com, http://docs.unity3d.com/ScriptReference/Vector3.Lerp.html [Zugriff am 09.12.2015]

```
public class LerpExample : MonoBehaviour {
    float lerpTime = 1f;
    float currentLerpTime;

    float moveDistance = 10f;

    Vector3 startPos;
    Vector3 endPos;

    protected void Start() {
        startPos = transform.position;
        endPos = transform.position + transform.up * moveDistance;
    }

    protected void Update() {
        //reset when we press spacebar
        if (Input.GetKeyDown(KeyCode.Space)) {
            currentLerpTime = 0f;
        }

        //increment timer once per frame
        currentLerpTime += Time.deltaTime;
        if (currentLerpTime > lerpTime) {
            currentLerpTime = lerpTime;
        }

        //lerp!
        float perc = currentLerpTime / lerpTime;
        transform.position = Vector3.Lerp(startPos, endPos, perc);
    }
}
```

Abbildung 10 - Nutzung der Lerp-Funktion[47]

Hierbei wird ein Timer mit jedem Frame um eins weiter gesetzt, wodurch sich das Objekt mit einer gleichbleibenden Geschwindigkeit auf sein Ziel zubewegt. Soll sich das Objekt jedoch mit veränderbarer Geschwindigkeit bewegen, muss der Wert t dahingehend angepasst werden.

[47] Utter, Robert (2014): *How to Lerp like a pro*, veröff. auf chicounity3d.wordpress.com, https://chicounity3d.wordpress.com/2014/05/23/how-to-lerp-like-a-pro/ [Zugriff am 09.12.2015]

Soll sich das Objekt immer langsamer bewegen, je näher es am Ziel ist (wie die Kamera in „Ori and the Blind Forest“), kann die folgende Sinusfunktion verwendet werden.

$$t = \sin\left(t * \frac{\pi}{2}\right)$$

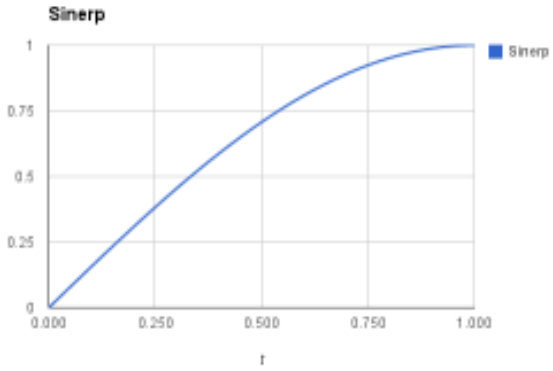


Abbildung 11 - Lerping mittels Sinusfunktion[48]

Umgekehrt kann das Objekt sich zunächst langsam und dann immer schneller werdend bewegen wenn die Cosinusfunktion verwendet wird.

$$t = \cos\left(t * \frac{\pi}{2}\right)$$

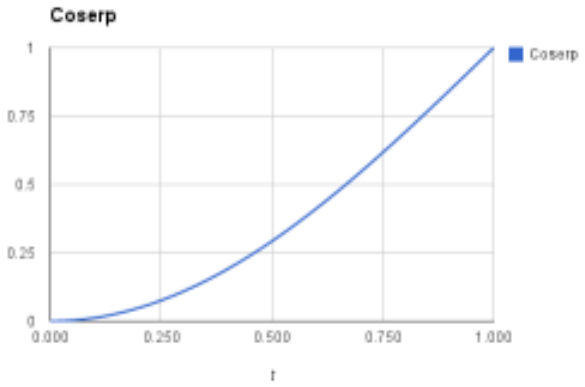


Abbildung 12 - Lerping mittels Kosinusfunktion[49]

[48] Utter, Robert (2014): *How to Lerp like a pro*, veröff. auf chicounity3d.wordpress.com, https://chicounity3d.wordpress.com/2014/05/23/how-to-lerp-like-a-pro/interp-sinerp.png [Zugriff am 09.12.2015]

Für eine geschmeidige Bewegung, welche sich sowohl zu Beginn als auch zum Ende der Bewegung langsam verhält, ist die Smoothstepfunktion[50] eine Option.

$$t = t * t * (3 - 2 * t)$$

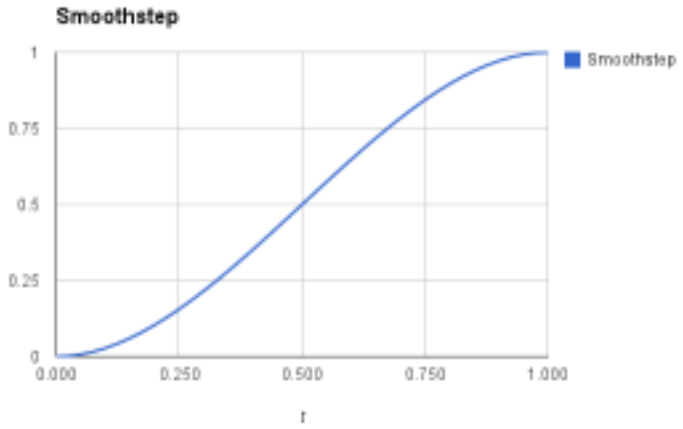


Abbildung 13 - Lerping mittels der Smoothstep-Funktion[51]

Auf diese Weise kann sich die Kamera hinter dem Spieler her bewegen, ohne dass es zu abrupten Bewegungen kommt.

In der Praxis hat sich herausgestellt, dass das Lerping hin zu bewegten Zielen ein ruckeliges Verhalten verursacht. Dieses Problem wurde von der Unity Community gelöst, indem eine eigene Lerp-Funktion entwickelt wurde, welche den Standort des Zieles im vorherigen Frame mit in die Berechnung einbezieht.[52]

---

[49] Utter, Robert (2014): *How to Lerp like a pro*, veröff. auf chicounity3d.wordpress.com, https://chicounity3d.wordpress.com/2014/05/23/how-to-lerp-like-a-pro/interp-coserp.png [Zugriff am 09.12.2015]

[50] Kesson, Malcolm (2002): *Using smoothstep*, veröff. auf fundza.com, http://www.fundza.com/rman_shaders/smoothstep/index.html [Zugriff am 09.12.2015]

[51] Utter, Robert (2014): *How to Lerp like a pro*, veröff. auf chicounity3d.wordpress.com, https://chicounity3d.wordpress.com/2014/05/23/how-to-lerp-like-a-pro/interp-smooth.png [Zugriff am 09.12.2015]

[52] CarlEmail (2012): *How to smooth damp towards a moving target without causing jitter in the movement?*, veröff. auf forum.unity3d.com, http://forum.unity3d.com/threads/how-to-smooth-damp-towards-a-moving-target-without-causing-jitter-in-the-movement.130920/#post-885121 [Zugriff am 09.12.2015]

3.1.3 Begründung der gewählten Variante

Die Methode des Kamerafensters ist einer der ältesten Versuche, die Kamera für den Spieler angenehmer zu gestalten. Weder in den untersuchten Beispielen im Kapitel 2.5 noch in anderen aktuellen Spielen wurden Beispiele für eine Nutzung eines Kamerafensters gefunden. Im Rahmen dieser Arbeit soll die Methode des Lerpings implementiert werden, da diese zeitgemäßer ist. Das Kamerafenster hat in sowohl Spielen mit gewünschtem Retro-Aspekt als auch in anderen Nischen seine Daseinsberechtigung. Da das zu implementierende System allgemein aufgebaut werden soll, wird an dieser Stelle auf das Kamerafenster verzichtet.

3.2 Parallax Scrolling

3.2.1 Bewegen des Hintergrundes

Wenn ein scrollender Hintergrund im Spiel einfach nur endlos durchlaufen soll ist es am einfachsten, die Hintergrundtexturen relativ zur Geschwindigkeit des Spielers zu bewegen und wiederkehrend einzufügen. Dazu muss nichts weiter getan werden als die Bewegungsgeschwindigkeit an die Textur zu übergeben. Diese wird dann mit einem Faktor zwischen 0 und 1 versehen, damit sie sich langsamer bewegt als der Spieler. Im Fall, dass mehrere verschiedene Hintergründe in mehreren Tiefenebenen verwendet werden sollen, wird jeder dieser Texturen ein anderer Faktor gegeben, absteigend je weiter die Textur im Hintergrund liegt.

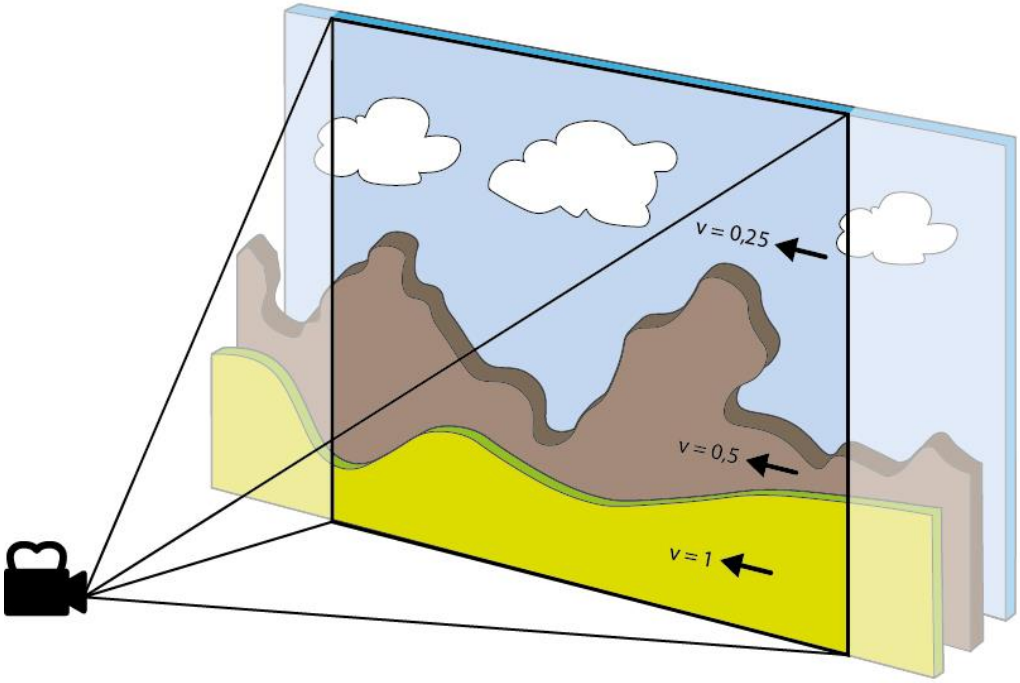


Abbildung 14 - Parallaxscrolling mittels bewegter Hintergründe

Diese Technik ist nützlich, um beispielsweise Himmel mit Wolken oder Berge zu erstellen. Wenn jedoch der Hintergrund genau zum Vordergrund passen soll, wie zum Beispiel im Spiel „Mutant Mudds“ (siehe 2.5.1), ist diese Technik schwer anzuwenden, da der Designer nicht auf einfache Weise vorausplanen kann wo im Level der entsprechende Abschnitt der scrollenden Textur sein wird.

3.2.2 Rendering mittels mehrerer Kameras

Wenn es in einem Spiel wichtig ist, dass die Parallaxebenen zum Vordergrund passen ist es für Designer einfacher die entsprechenden Teile der Ebene als separate Assets zu erstellen und diese zu platzieren. In großen Levels kann dies schnell zu einer großen Menge an Assets führen. Diese können zwar als Gruppe zusammengefasst und so bewegt werden, jedoch führt dies dazu, dass große Mengen an Objekten bewegt werden müssen, welches die Performanz des Spiels einschränken kann. Eine andere Möglichkeit ist, stattdessen eine Kamera pro Ebene zu haben, welche nur diese rendert. Anstelle des Hintergrundes wird dann die jeweilige Kamera bewegt. Analog zu

der Lösung mit bewegtem Hintergrund werden die Kameras mit unterschiedlicher Geschwindigkeit bewegt, um den gewünschten Tiefeneffekt zu erzielen.

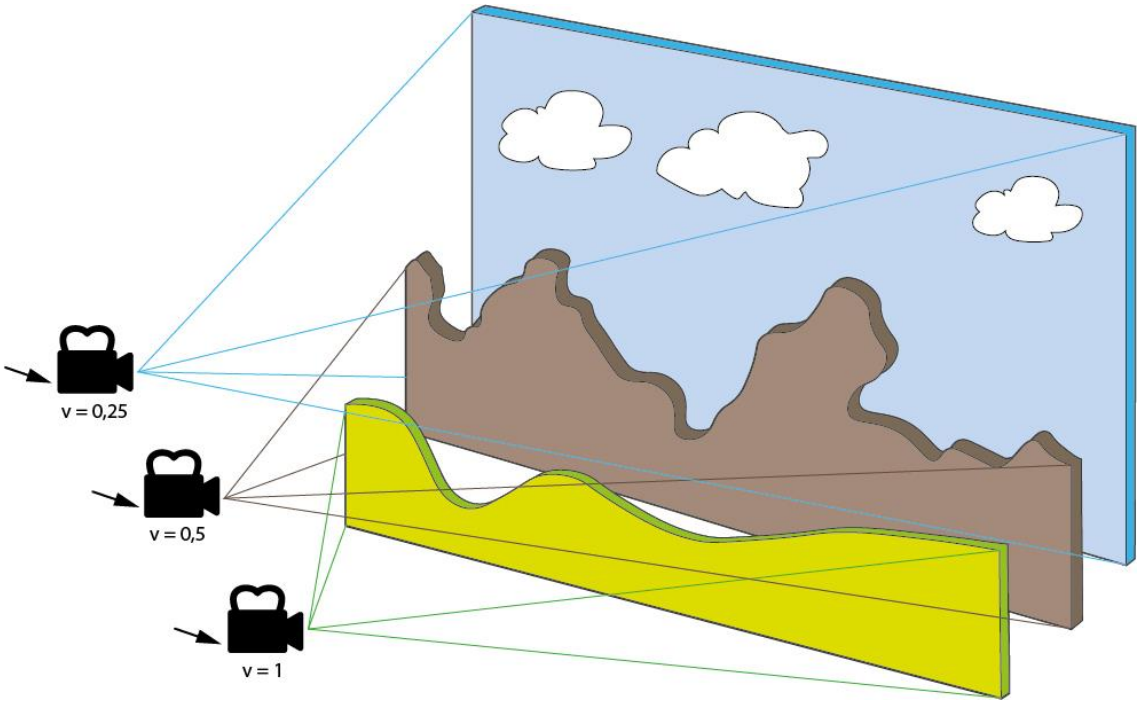


Abbildung 15 - Parallaxscrolling mittels bewegter Kameras

3.2.3 Begründung der gewählten Variante

Beide Versionen haben ihre Vorteile. Um ein Ergebnis bezüglich der Wahl zu erzielen werden die jeweiligen Vorteile und Nachteile gegenübergestellt.

Tabelle 2 - Vor- und Nachteile von bewegten Hintergründen und bewegten Kameras

Bewegen des Hintergrundes	**Mehrere bewegte Kameras**
Vorteile	**Vorteile**
Einfache Umsetzung endloser Hintergründe	Spezielle an Vordergrund angepasste Hintergründe lassen sich einfacher umsetzen
Einfache Implementierung	Modulare Hintergrundsets können direkt in der Engine umgesetzt werden
Nachteile	**Nachteile**
Ungeeignet, falls die Kamera zoomen soll	Komplexere Implementierung
komplexe, zusammengesetzte Hintergründe können zu Einschränkungen der Performanz führen	

Aus der Tabelle lässt sich ablesen, dass beide Systeme für jeweils eine spezifische Art von Spielen besser geeignet ist. In einem Endless Runner mit einfachem Hintergrund ist die erste Methode besser, für komplex abgestimmte Hintergrundebenen jedoch die zweite. Auch abseits dieser spielabhängigen Vorteile gibt es Unterschiede. So lässt sich Methode 1 mit deutlich weniger Aufwand implementieren, allerdings kann sie bei einer Zoomfunktion den Tiefeneffekt im Hintergrund nicht vernünftig simulieren, da dieser dafür skaliert werden müsste.

Implementierungsaufwand soll nicht als entscheidendes Kriterium gelten, da die Priorität darin liegt, dem Nutzer möglichst viel Arbeit abzunehmen, nicht notwendigerweise dem Ersteller des Systems. Da der Designer die Möglichkeit haben soll in seinem Spiel Zoomfunktionen zu implementieren wird sich für die zweite Variante, das Rendering mittels mehrerer Kameras, entschieden.

3.3 Implementierung der Kamera durch Designer

In diesem Abschnitt wird gegenübergestellt, auf welche Arten genau der Game Designer die Kamera schlussendlich ins Spiel integriert und wie er Kontrolle über ihr Verhalten bekommt. Dies kann zum einen durch ein zentrales Interface geschehen, in welchem er das Verhalten auswählt, oder zum anderen durch das Setzen visueller Elemente im Level-Editor der jeweiligen Engine.

3.3.1 Zentrales Benutzer-Interface

Da es das Anliegen dieser Arbeit ist ein Tool zu entwickeln, welches von Designern ohne weitere Programmierkenntnisse verwendet werden kann, sollten diese so wenig wie möglich mit dem Programmcode in Berührung kommen. Daher muss ihnen eine Möglichkeit gegeben werden das Programm ohne die Eingabe von Quellcode zu bedienen. Dies ist vergleichbar mit der Entwicklung von textbasierten Schnittstellen bis hin zu grafischen User Interfaces (GUI), welche Anwendungsprogramme einer breiteren Masse zugänglich machten.[53]

Eine gut designte grafische Oberfläche bringt dabei folgende Vorteile mit sich.[54]

1. Bedienbarkeit unabhängig vom Wissens- oder Fähigkeitsstand des Nutzers
2. Beschleunigung der Arbeitsprozesse
3. Erleichterung des Einstiegs in die Arbeit
4. Fokussierung auf den Arbeitsprozess durch Weglassen ablenkender Elemente
5. Einsparen unnötiger Erklärungen oder Hinweise

[53] Vgl. ITWissen (Hrsg.) (2015): *Benutzeroberfläche*, veröff. auf itwissen.info, http://www.itwissen.info/definition/lexikon/Benutzeroberflaeche-UI-user-interface.html [Zugriff am 09.12.2015]

[54] Vgl. Semmler, Jan (2012): *Vorteile eines gut designten grafischen User-Interfaces*, veröff. auf jansemmler.de, http://www.jansemler.de/interface/vorteile-eines-gut-designten-grafischen-user-interfaces/ [Zugriff am 09.12.2015]

Um diese Vorteile optimal nutzen zu können sollte das Programm aus einem einzigen Interface bestehen, welches einfach zu finden ist. Dadurch kann der Designer sämtliche Einstellungen an einem einzigen Ort ändern und muss nicht an verschiedenen Stellen Änderungen vornehmen, um das Ergebnis zu testen.

Die Analyse bestehender Sidescroller-Spiele hat ergeben, dass in einigen Spielen einfache Regeln für die Kameraverfolgung genügen. Solche lassen sich einfach in einem Interface aktivieren oder deaktivieren. Auch Einstellungen, welche einen Wert benötigen (z.B. Verfolgungsgeschwindigkeit) lassen sich gut über ein zentrales Interface steuern.

Gerade in Spielen wie „The Cave“, „Never Alone“ (siehe Kapitel 2.5.3 und 2.5.4) wird ersichtlich, dass einfache Einstellungsmöglichkeiten allein nicht für alle Spiele ausreichend sind. Diese Spiele haben Kameras, welche besonderen Regeln folgen. Diese Regeln sind nicht eins zu eins in anderen Spielen wiederzufinden. Um solch komplexe Kameras zu implementieren müsste das Interface stark an die Bedürfnisse des entsprechenden Spiels angepasst werden. Da jedoch ein allgemeingültiges Tool entwickelt werden soll, kann ein Interface nicht alle möglichen Varianten beinhalten und gleichzeitig übersichtlich und einfach zu bedienen sein. Aus diesem Grund stellt Maximilian Csuk eine weitere Methode vor um die Kamera zu führen, er nennt diese „In-level camera helpers“.[55]

3.3.2 Individuelle visuelle Elemente

In Fällen in denen es nicht möglich ist eine Kamerabewegung mittels allgemeingültiger Regeln zu erreichen, können Hilfselemente verwendet werden, welche im Level individuell platziert werden können. Dabei fungieren diese Elemente als Auslöser, welche zu einem bestimmten Grad die Kamera beeinflussen. Dies kann zum Beispiel einfach nur bedeuten, dass die Kamera zu dem Hilfselement wie einem Attraktor angezogen wird. Generell können aber alle möglichen Kameraverhalten durch ein solches Hilfselement ausgelöst werden. Ein gutes Beispiel dafür sind Zooms, wie in

[55] Csuk, Maximilian (2012): *Cameras in 2D platformers*, veröff. auf imake-games.com, http://www.imake-games.com/cameras-in-2d-platformers/ [Zugriff am 09.12.2015]

„The Cave“, welche ausgelöst werden wenn ein Spieler einen bestimmten Abschnitt betritt.

Attraktoren können auf vielfältige Weise von Level Designern eingesetzt werden. Sie können die Kamera auf gewisse Elemente lenken, welche wichtig für den Spielverlauf sind. Dies sind zum Beispiel Schlüssel, die zum Öffnen von Türen benötigt werden. Andererseits können sie die Kamera aber auch von Elementen ablenken, wie zum Beispiel durch den Spieler nur schwer auffindbare geheime Eingänge. Somit bieten sie genau das, was für die Erfüllung des Punktes „Designerkontrolle“ (siehe 2.2.2) benötigt wird. Des Weiteren können Sie im Editor als einzelne Elemente gesetzt werden, wodurch die Designer deren Position auf einfache Weise solange ändern können bis sie zufrieden sind. Bei der Implementierung muss dabei nur bedacht werden, dass diese Hilfselemente zwar im Editor visuell dargestellt werden, aber im Spiel selbst nicht sichtbar sein dürfen.

Ein großer Nachteil solch einzelner Elemente ist, dass Sie alle separat im Level platziert werden müssen. Selbst bei einer gut strukturierten Level-Hierarchie kann dies schnell unübersichtlich werden, wenn bestimmte Elemente später einmal wieder editiert werden sollen. Somit bringt diese Funktionalität leider eine Einschränkung der Bedienbarkeit mit sich.

3.3.3 Begründung der gewählten Variante

Um auf der einen Seite eine optimale Bedienbarkeit und auf der anderen Seite eine gute Funktionalität zu gewährleisten, sollen die beiden aufgezeigten Möglichkeiten kombiniert werden. Das Programm soll über ein zentrales GUI gesteuert werden, in welchem die grundlegenden Verhaltensmuster der Kamera eingestellt werden. Außerdem werden von diesem Interface aus einzelne Kamerahelfern erstellt und es kann zu ihnen navigiert werden. Die Benutzeroberfläche dient als zentrales Element für den Designer, welches ihm die Verwendung vereinfacht.

4 Beschreibung der Lösung

4.1 Kontextsicht

Die Kontextsicht beschreibt die Einbettung des Systems SidescrollerCamera in seine Umgebung, das heißt die Interaktion mit externen Elementen. Das System selbst wird hier als Blackbox betrachtet. Die externen Elemente sind hierbei die Unity3D Engine und der Entwickler, welcher das System nutzt um Spiele zu entwickeln. Das System erbt von Unitys MonoBehaviour-Klasse und steuert Unitys Kamera.

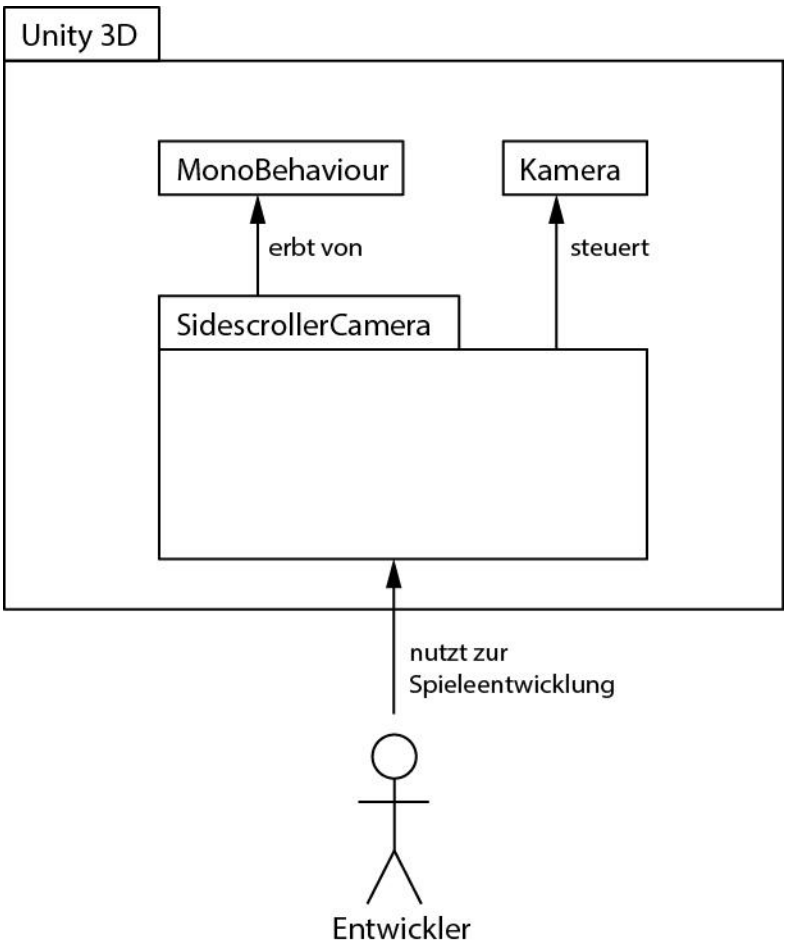


Abbildung 16 - Kontextsicht

4.2 Struktursicht

Die Struktursicht zeigt die Komponenten bzw. Module des Systems SidescrollerCamera und deren Zusammenspiel sozusagen aus der Vogelperspektive. Die Komponenten stellen die Grundpfeiler der Software dar. Darüber hinaus wurde auf die Vererbungsbeziehung aus der Unity3D Engine eingegangen. Das hier gezeigte Objektdiagramm dient der einfachen Übersicht über das System, ein komplettes Klassendiagramm ist dem Anhang zu entnehmen.

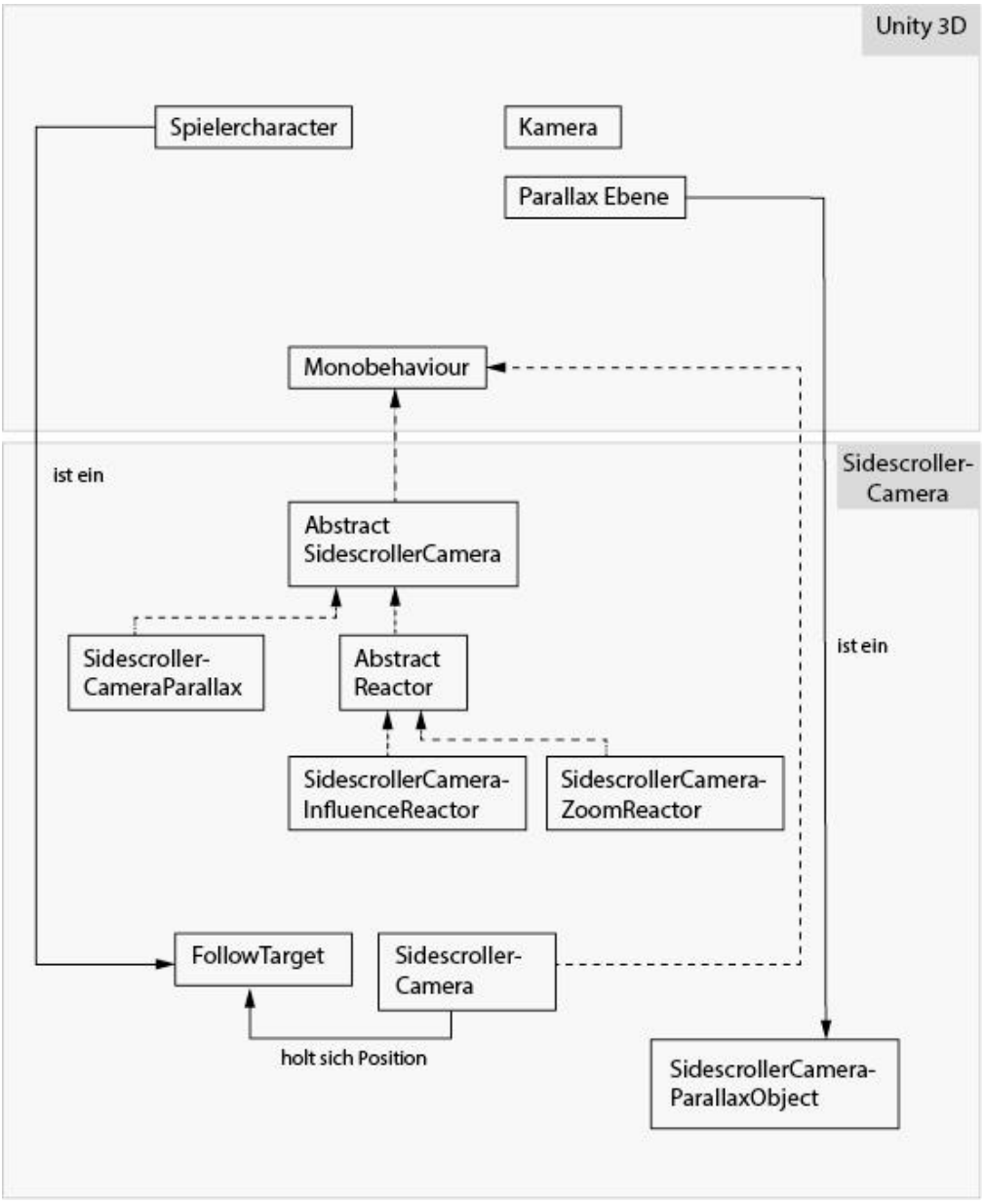


Abbildung 17 - Objektdiagramm

4.3 Verhaltenssicht

In der Verhaltenssicht wird dargestellt, in welcher zeitlichen Abfolge die verschiedenen Komponenten miteinander interagieren. Zu diesem Zweck wurde ein Sequenzdiagramm angefertigt, welches die Berechnung der Kameraverfolgung darstellt. Dabei wird auf den möglichen Einfluss von Reaktoren eingegangen.

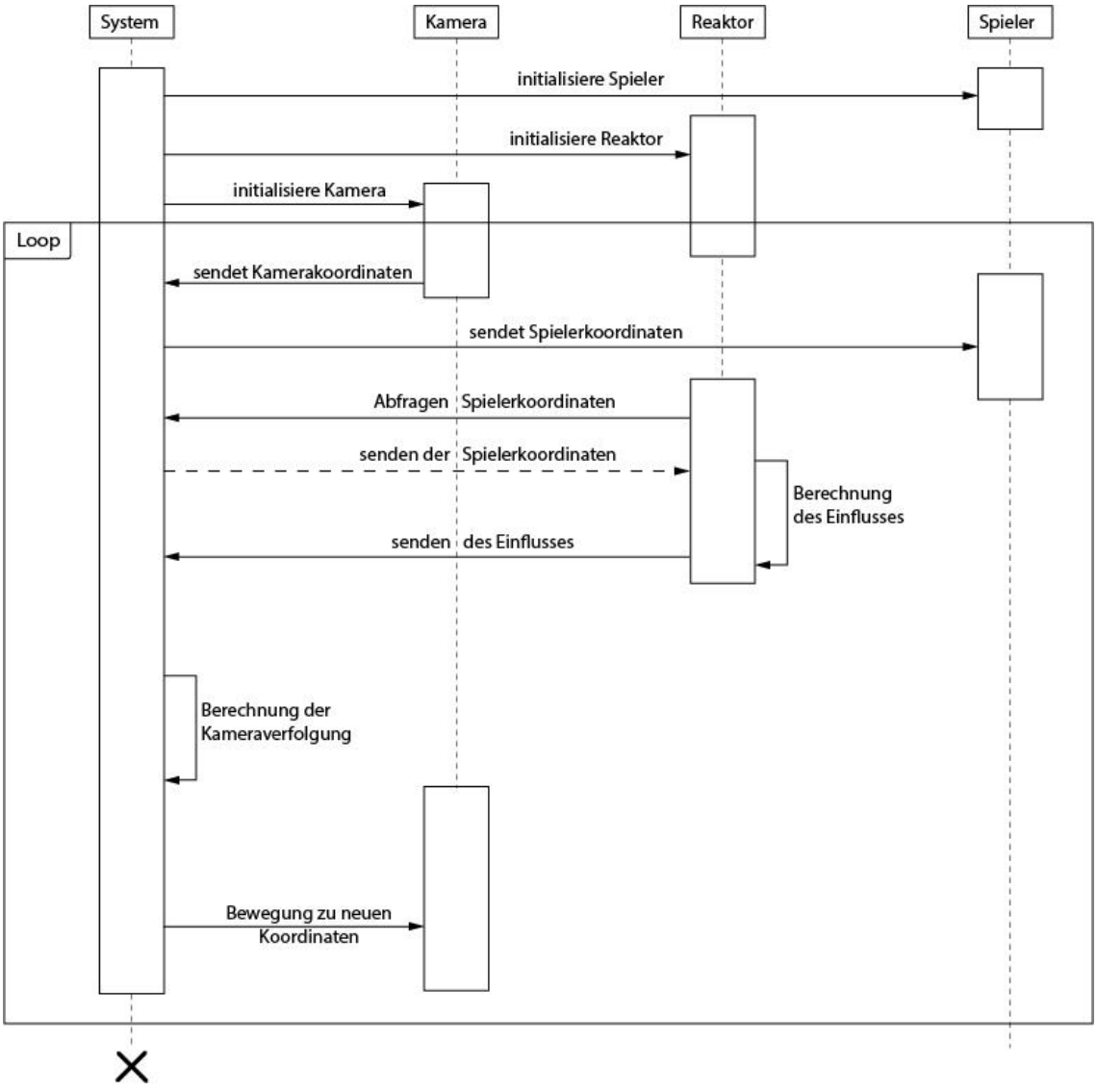


Abbildung 18 - Sequenzdiagramm

4.4 Komponenten und Subsysteme

4.4.1 Übergreifende Aspekte, Infrastruktur, Schnittstellen

Das Bachelorprojekt wird unter Zuhilfenahme der Unity3D Engine in der Version 5.5.2f1 (64 bit) umgesetzt. Diese stellte die Basisklasse MonoBehaviour bereit, auf welche während der Entwicklung zurückgegriffen wurde.

Als Programmierschnittstelle werden unter der Unity3D Engine Javascript und C# angeboten. Die Entscheidung fiel hier aufgrund der größeren Erfahrung mit dieser Programmiersprache auf C#. Als Entwicklungsumgebung diente die Community Edition von Visual Studio 2015.

Entwickelt wurde das Projekt auf einem Windows PC mit folgenden Spezifikation:

Prozessor: Intel Core i7-4700MQ @ 2,4 GHz mit 4 Kernen

Arbeitsspeicher: 8 GB DDR 3 RAM

Grafikkarte: NVIDIA GeForce GTX 765M

Unity gibt auf der firmeneigenen Website an, dass für die Entwicklung von Spielen mit ihrer Engine Grafikkarten mit DirectX9 und Shadermodel 2.0 nötig sind. Restliche Spezifikationen hängen von den Spielen ab, welche entwickelt werden.[56]

Für das Speichern von Arrays im Unity Editor wurde die freie Klasse EditorPrefsX von Eric Haines verwendet.[57]

---

[56] Unity rechnologies (Hrsg.) (2015): *System Requirements for Unity 5.3*, veröff. auf uni-ty3d.com, https://unity3d.com/unity/system-requirements [Zugriff am 09.12.2015]

[57] Haines, Eric (2014): *ArrayPrefs2*, veröff. auf wiki.unity3d.com, http://wiki.unity3d.com/index.php/ArrayPrefs2 [Zugriff am 09.12.2015]

4.4.2 SidescrollerCamera

Dieses Modul des Plugins ist für die hauptsächliche Kameraverfolgung zuständig. Dafür wurde das bereits beschriebene Lerping implementiert und so strukturiert, dass der Nutzer möglichst viele Einstellungsmöglichkeiten hat.

Mittelpunkt mehrerer Ziele

Da der Designer die Möglichkeit haben soll die Kamera mehreren Zielen gleichzeitig folgen zu lassen, soll die Kamera sich auf den Mittelpunkt der Ziele fokussieren. Hierzu wird jedoch nicht einfach der normale Mittelpunkt, sondern ein gewichteter Mittelpunkt genommen. Dies erlaubt es dem Designer, ein oder mehrere Ziele „wichtiger“ zu machen als andere.

```
//midpoint between all targets, including their weights
Vector3 GetTargetsWeightedMidPoint(IList<FollowTarget> targets)
{
    var midPointH = 0f;
    var midPointW = 0f;
    var totalTargets = targets.Count;

    if (totalTargets == 0)
        return transform.localPosition;

    var totalInfluencesH = 0f;
    var totalInfluencesW = 0f;
    var totalAccountableTargetsH = 0;
    var totalAccountableTargetsV = 0;
    for (int i = 0; i < totalTargets; i++)
    {
        if (targets[i] == null)
            continue;

        midPointH += (Vector3H(targets[i].TargetPosition) + targets[i].TargetOffset.x) * targets[i].TargetInfluenceH;
        midPointW += (Vector3W(targets[i].TargetPosition) + targets[i].TargetOffset.y) * targets[i].TargetInfluenceV;

        totalInfluencesH += targets[i].TargetInfluenceH;
        totalInfluencesW += targets[i].TargetInfluenceV;

        if (targets[i].TargetInfluenceH > 0)
            totalAccountableTargetsH++;

        if (targets[i].TargetInfluenceV > 0)
            totalAccountableTargetsV++;
    }

    if (totalInfluencesH < 1 && totalAccountableTargetsH == 1)
        totalInfluencesH += (1 - totalInfluencesH);

    if (totalInfluencesW < 1 && totalAccountableTargetsV == 1)
        totalInfluencesW += (1 - totalInfluencesW);

    if (totalInfluencesH > .0001f)
        midPointH /= totalInfluencesH;

    if (totalInfluencesW > .0001f)
        midPointW /= totalInfluencesW;

    return VectorHW(midPointH, midPointW);
}
```

Abbildung 19 - Ermittlung des gewichteten Mittelpunktes

Hierfür werden mittels einer Schleife sämtliche Koordinaten addiert, nachdem ihr Einfluss aufgerechnet wurde. Das Ergebnis wird am Ende durch den Gesamteinfluss geteilt, wodurch der gewichtete Mittelpunkt errechnet ist.

Maximale Kameraentfernung

Da der Designer die Möglichkeit hat die maximale Entfernung der Kamera zum Spieler festzulegen, muss beachtet werden was passiert, sollte die Kamera zu langsam sein. In diesem Fall wird zusätzlich zu der zuvor in der Methode Move() berechneten Bewegung der Betrag der Überschreitung der Entfernungsgrenze addiert.

Hier stellte sich die Herausforderung, das für die Kamera, den Bildschirm des Spielers und das Level unterschiedliche Koordinatensysteme verwendet werden. Die *World Coordinates* beschreiben die Position eines Objekts im Level. Die Einheit für dieses Koordinatensystem wird *Unit* genannt. *View Coordinates* werden dafür verwendet, die Position von Objekten im Viewport der Kamera anzugeben. Dabei sind die Koordinaten der unteren linken Ecke [0,0] und der oberen rechten Ecke [1,1]. Die tatsächlichen
Pixelkoordinaten auf dem Ausgabegerät werden *Screen Coordinates* genannt. Da die maximale Kameraentfernung in View Coordinates angegeben wird, die Berechnung der Bewegung jedoch in World Coordinates geschieht, muss hier umgerechnet werden.[58]

```
// Limit camera distance to target by adding the amount the boundary has been overstepped to the position after tweening
if (LimitHorizontalCameraDistance)
{
    var horizontalCompensation = Vector3H(_FollowTargetPosition) - Vector3H(_transform.localPosition) - (ScreenSizeInWorldCoordinates.x / 2)
        * MaxHorizontalTargetDistance;
    if (horizontalCompensation > 0)
    {
        _FollowTargetHorizontalPositionSmoothed += horizontalCompensation;
        _previousFollowTargetHorizontalPositionSmoothed = _FollowTargetHorizontalPositionSmoothed;
    }
    else if (horizontalCompensation < -ScreenSizeInWorldCoordinates.x * MaxHorizontalTargetDistance)
    {
        _FollowTargetHorizontalPositionSmoothed += horizontalCompensation + ScreenSizeInWorldCoordinates.x * MaxHorizontalTargetDistance;
        _previousFollowTargetHorizontalPositionSmoothed = _FollowTargetHorizontalPositionSmoothed;
    }
}
```

Abbildung 20 - Umrechnung von View Coordinates in World Coordinates

[58] Vgl. Sonofrage (2014): *Working with the coordinate system and game screen in Unity 2DK,* veröff. auf stackoverflow.com, http://stackoverflow.com/questions/21937544/working-with-the-coordinate-system-and-game-screen-in-unity-2d [Zugriff am 09.12.20015]

Updatezyklen

Da Unity mehrere Updatezyklen hat, kann die Bewegung der Kamera auch in unterschiedlichen Zyklen durchgeführt werden. Die Wahl des richtigen Zyklus hängt dabei von der Bewegung der verfolgten Objekte ab. Objekte, welche sich physikalisch korrekt verhalten sollen, werden im FixedUpdate-Loop bewegt. Dies geschieht, da dieser Loop unabhängig von der Framerate ist, auf welcher die Anwendung läuft. Die Kamera muss in diesem Fall auch im FixedUpdate-Loop laufen. Es kann sonst zu ruckeligem Verhalten kommen kann, wenn der FixedUpdate-Loop häufiger oder seltener aufgerufen wird als die anderen Update-Funktionen. In jedem anderen Beispiel ist es ratsam die Kamera im LateUpdate-Zyklus zu bewegen. Dieser ist der letzte aller Update-Zyklen. Wenn die Kamera nicht einem physikalischen Objekt folgt, sollte sie auch nicht im FixedUpdate-Zyklus laufen, da dieser öfter aufgerufen wird und somit die Leistungsfähigkeit der Anwendung leiden könnte.[59]

```
/// <summary>
/// camera gets moved in the late update cycle if chosen
/// </summary>
void LateUpdate()
{
    if (Enabled && UpdateCycle == UpdateCycle.LateUpdate)
        Move();
}

/// <summary>
/// camera gets moved in the fixed update cycle if chosen
/// </summary>
void FixedUpdate()
{
    if (Enabled && UpdateCycle == UpdateCycle.FixedUpdate)
        Move();
}
```

Abbildung 21 - Methodenaufruf in verschiedenen Update-Zyklen

[59] Vgl. Unity Technologies (Hrsg.) (2015): *Execution Order,* veröff. auf docs.unity3d.com, http://docs.unity3d.com/Manual/ExecutionOrder.html [Zugriff am 09.12.2015]

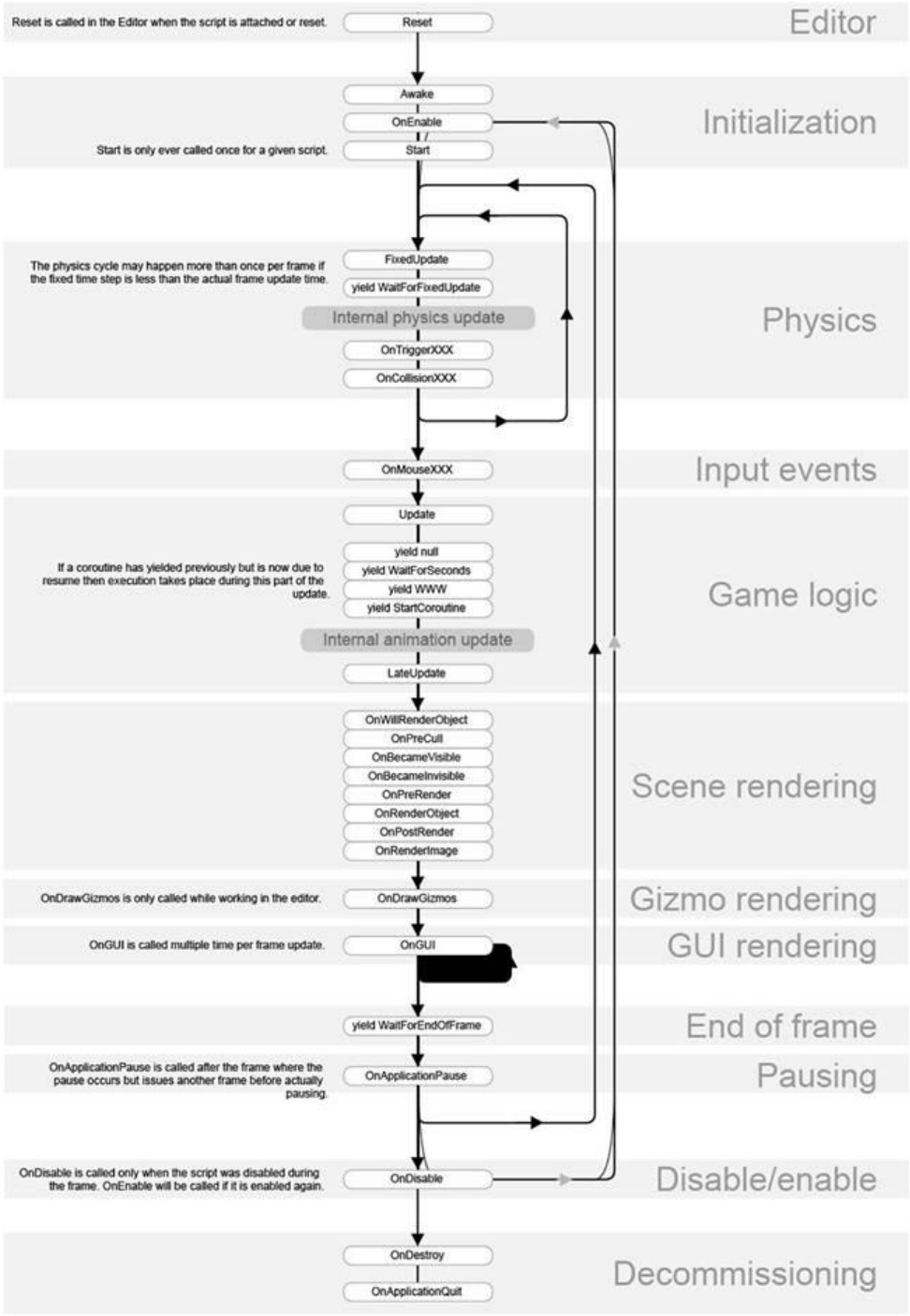


Abbildung 22 - Unity execution order[60]

[60] Vgl. Unity Technologies (Hrsg.) (2015):
http://docs.unity3d.com/uploads/Main/monobehaviour_flowchart.svg [Zugriff am 09.12.2015]

4.4.3 Reactors

Um dem Designer mehr Kontrolle über den Fokus der Kamera zu geben werden Elemente implementiert, welche es ihm erlauben die Kamera zu beeinflussen, sollte der Spieler einen bestimmten Bereich betreten. Dies geschieht nach dem Vorbild von „The Cave“ (siehe Kapitel 2.5.3). Dafür werden zwei verschiedene Reaktoren benötigt, einer für die Verlagerung des Kamerafokus und einer für den Zoom. Da beide jedoch viele gemeinsame Methoden haben wird eine abstrakte Klasse erstellt, von der diese beiden erben.

AbstractReactors

Die AbstractReactors-Klasse beinhaltet sämtliche Funktionen, welche sowohl für den ZoomReactor als auch den InfluenceReactor wichtig sind. Dies sind vor allem die Überprüfung, ob sich das Verfolgungsziel der Kamera innerhalb eines Triggers befindet, und zu welchem prozentualen Anteil sich das Verfolgungsziel zum Mittelpunkt des Triggers befindet.

Um den Triggern mehr Variabilität zu geben wurden diese in zwei Teile untergliedert. Im prozentualen Teil des Triggers ist der jeweilige Einfluss des Reaktors auf die Kamera davon abhängig, wie weit sich der Spieler genau innerhalb des Triggers befindet. Im exklusiven Teil des Triggers ist der Einfluss des Reaktors vollständig.

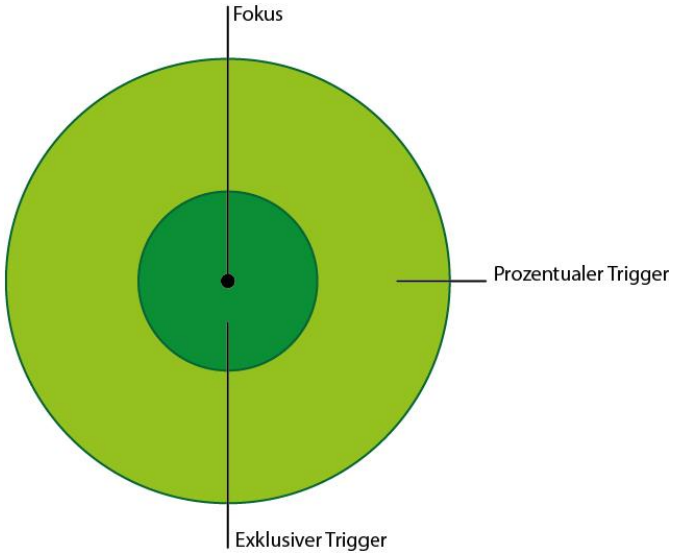


Abbildung 23 - Triggeraufbau

```
//calculating distance between target and trigger center in percentages
protected float GetDistanceToCenterPercentage(Vector2 point)
{
    _vectorFromPointToCenter = point - new Vector2(Vector3H(_transform.position), Vector3W(_transform.position));
    if (TriggerShape == TriggerShape.RECTANGLE)
    {
        var distancePercentageH = Vector3H(_vectorFromPointToCenter) / (Vector3H(_transform.localScale) * .5f);
        var distancePercentageV = Vector3W(_vectorFromPointToCenter) / (Vector3W(_transform.localScale) * .5f);
        var distancePercentage = (Mathf.Max(Mathf.Abs(distancePercentageH),
            Mathf.Abs(distancePercentageV))).Remap(_exclusiveInfluencePercentage, 1, 0, 1);
        return distancePercentage;
    }
    else
    {
        var distancePercentage = (_vectorFromPointToCenter.magnitude / ((Vector3H(_transform.localScale) +
            Vector3W(_transform.localScale)) * .25f)).Remap(_exclusiveInfluencePercentage, 1, 0, 1);
        return distancePercentage;
    }
}
```

Abbildung 24 - Berechnung der Distanz zum Triggermittelpunkt

```
IEnumerator TestTriggerRoutine()
{
    yield return new WaitForEndOfFrame();

    var waitForSeconds = new WaitForSeconds(UpdateInterval);
    while (true)
    {
        var triggerPos = SidescrollerCamera.TargetsMidPoint;
        if (!UseTargetsMidPoint && TriggerTarget != null)
            triggerPos = TriggerTarget.position;

        if (TriggerShape == TriggerShape.RECTANGLE &&
            IsInsideRectangle(
                Vector3H(_transform.position),
                Vector3W(_transform.position),
                Vector3H(_transform.localScale),
                Vector3W(_transform.localScale),
                Vector3H(triggerPos),
                Vector3W(triggerPos)))
        {
            if (!_insideTrigger)
                EnteredTrigger();
        }
        else if (TriggerShape == TriggerShape.CIRCLE &&
                 IsInsideCircle(
                Vector3H(_transform.position),
                Vector3W(_transform.position),
                (Vector3H(_transform.localScale) + Vector3W(_transform.localScale)) * .25f,
                Vector3H(triggerPos),
                Vector3W(triggerPos)))
        {
            if (!_insideTrigger)
                EnteredTrigger();
        }
        else
        {
            if (_insideTrigger)
                ExitedTrigger();
        }
        yield return waitForSeconds;
    }
}
```

Abbildung 25 - Ermittlung ob Spieler im Trigger ist

Exemplarisch für alle möglichen Reaktoren soll an dieser Stelle näher auf den InfluenceReactor eingegangen werden. Das Herzstück dessen ist die Beeinflussung der Kamera, wenn das Verfolgungsziel sich innerhalb des Triggers befindet. Hierbei wird unterschieden, welcher Teil des Triggers aktiv ist. Dementsprechend wird entweder mit einer Distanz zum Mittelpunkt von 0 Prozent oder einem höheren Prozentsatz gerechnet.

```
//when target is inside the trigger
IEnumerator InsideTriggerRoutine()
{
    var waitForFixedUpdate = new WaitForFixedUpdate();
    yield return (SidescrollerCamera.UpdateCycle == UpdateCycle.FixedUpdate) ? waitForFixedUpdate : null;

    _tempExclusivePoint = new Vector2(Vector3H(SidescrollerCamera.FollowTargetPosition),
        Vector3W(SidescrollerCamera.FollowTargetPosition));
    while (_insideTrigger)
    {
        _exclusiveInfluencePercentage = ExclusiveInfluencePercentage;

        var distancePercentage = GetDistanceToCenterPercentage(new Vector2(Vector3H(SidescrollerCamera.TargetsMidPoint),
            Vector3W(SidescrollerCamera.TargetsMidPoint)));
        var vectorFromPointToFocus = new Vector2(Vector3H(SidescrollerCamera.TargetsMidPoint),
            Vector3W(SidescrollerCamera.TargetsMidPoint)) - new Vector2(Vector3H(FocusPoint.position), Vector3W(FocusPoint.position));
        if (distancePercentage == 0)
        {
            //Focus is entirely on chosen focuspoint
            SidescrollerCamera.ExclusiveTargetPosition = Vector2.SmoothDamp(_tempExclusivePoint,
                new Vector2(Vector3H(FocusPoint.position), Vector3W(FocusPoint.position)), ref _velocity, InfluenceSmoothness);
            _tempExclusivePoint = SidescrollerCamera.ExclusiveTargetPosition.Value;
            _influence = -vectorFromPointToFocus * (1 - distancePercentage);
            SidescrollerCamera.UseInfluence(_influence);
        }
        else
        {
            //camera is influenced towards chosen focuspoint
            _influence = Vector2.SmoothDamp(_influence, -vectorFromPointToFocus * (1 - distancePercentage),
                ref _velocity, InfluenceSmoothness);
            SidescrollerCamera.UseInfluence(_influence);
            _tempExclusivePoint = new Vector2(Vector3H(SidescrollerCamera.FollowTargetPosition),
                Vector3W(SidescrollerCamera.FollowTargetPosition));
        }

        yield return (SidescrollerCamera.UpdateCycle == UpdateCycle.FixedUpdate) ? waitForFixedUpdate : null;
    }
}
```

Abbildung 26 - Funktionalität innerhalb des Triggers

4.4.4 Parallax

Das Erreichen eines Parallaxeffekts soll wie unter 3.2.2 aufgeführt mittels mehrerer Kameras geschehen, welche sich unterschiedlich schnell bewegen und jeweils nur eine Parallaxebene rendern. Das Bewegen der Kameras ist dabei verhältnismäßig leicht umzusetzen, da die Position jeder Ebene nur mit ihrer jeweiligen Geschwindigkeit multipliziert werden muss.

```
void Move()
{
    Vector3 rootOffset = transform.position - RootPosition;

    for (int i = 0; i < ParallaxLayers.Count; i++)
    {
        if (ParallaxLayers[i].CameraTransform != null)
        {
            // Position
            float x = ParallaxHorizontal ? Vector3H(rootOffset) * ParallaxLayers[i].Speed : Vector3H(rootOffset);
            float y = ParallaxVertical ? Vector3W(rootOffset) * ParallaxLayers[i].Speed : Vector3W(rootOffset);
            ParallaxLayers[i].CameraTransform.position = RootPosition + VectorHWD(x, y, Vector3D(transform.position));

            // Zoom
            ParallaxLayers[i].ParallaxCamera.orthographicSize = _initialOrtographicSize +
                (SidescrollerCamera.InGameCamera.orthographicSize - _initialOrtographicSize) * ParallaxLayers[i].Speed;
        }
    }
}
```

Abbildung 27 - Bewegung der Parallaxebenen

Um in jeder Kamera nur die ihr zugewiesene Ebene zu rendern gibt es in Unity die sogenannten Layer[61]. Diese können Objekte einem Oberbegriff zuordnen. Für den Parallaxeffekt bietet es sich an, die jeweiligen Layer nach einem Muster in Form von ParallaxLayerX zu benennen, wobei X eine aufsteigende ganzzahlig natürliche Zahl ist, beginnend mit der Zahl 1. Damit die entsprechende Kamera nur der jeweilige Layer rendert muss dieser in den Kameraoptionen unter *Culling Mask* ausgewählt werden.

[61] Unity Technologies (Hrsg.) (2015): *Layers,* veröff. auf docs.unity3d.com, http://docs.unity3d.com/Manual/Layers.html [Zugriff am 09.12.2015]

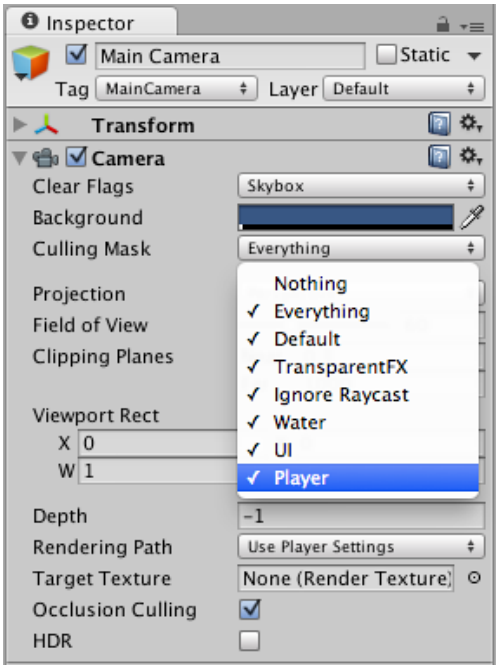


Abbildung 28 - Unitys Kameraoptionen[62]

[62] Vgl. Unity Technologies (Hrsg.) (2015): *http://docs.unity3d.com/uploads/Main/monobehaviour_flowchart.svg* [Zugriff am 09.12.2015]

5 Abschlussbetrachtung

5.1 Funktionsumfang des Prototypen

Der entwickelte Prototyp mit der Bezeichnung „Side Scroller Camera“ erfüllt in den aufgeführten Teilbereichen folgende Funktionen.

Tabelle 3 - Funktionsübersicht des Prototypen SidescrollerCamera

Modul	**Funktionen**
Parallax	• Das System simuliert einen Tiefeneffekt beim Bewegen der Spielfigur • Der Effekt wird über das Rendern der verschiedenen Ebenen durch verschiedene Kameras erreicht, welche sich mit unterschiedlicher Geschwindigkeit bewegen • Der Effekt wird auch beim Zoomen der Kamera erreicht • Der Designer kann die gewünschten Ebenen im Interface auswählen und deren Geschwindigkeit einstellen

Spielerverfolgung	• Die Kamera folgt dem Spieler
	• Der Entwickler hat die Möglichkeit, folgende Aspekte individuell einzustellen:
	  • Verfolgung auf einzelnen Achsen
	  • Verfolgung mehrerer Spieler mit individuellen Wichtungen
	  • Offset zwischen verfolgtem Spieler und Kamerafokus

	• Sanftheit der Verfolgung • Maximale Entfernung der Kamera vom Spieler • Maximale Geschwindigkeit der Kamera • Wahl zwischen verschiedenen Update-Zyklen • Die Kameraverfolgung wird mittels einer abgewandelten Funktion des Lerpings erreicht
InfluenceReactor	• Bewegt den Fokus der Kamera hin zu einem Fokuspunkt • Der äußere Triggerbereich bewegt den Fokus proportional zu der Position des Zieles im Verhältnis zum Triggermittelpunkt • Der innere Triggerbereich setzt den Fokus vollständig zum gewählten Fokuspunkt • Die Form des Reaktors kann elliptisch oder rechteckig sein • Der Fokuspunkt kann außerhalb des Triggers gewählt werden • Die Häufigkeit der Triggerüberprüfung ist anpassbar
ZoomReactor	• Verändert den Zoom der Kamera • Der äußere Triggerbereich zoomt die Kamera proportional zu der Position des Zieles im Verhältnis zum Triggermittelpunkt • Der innere Triggerbereich setzt den Zoom vollständig auf den gewählten neuen Kameraausschnitt • Der Zoom kann als Multiplikator oder als fester Zielwert eingestellt werden • Die Form des Reaktors kann elliptisch oder rechteckig sein

	• Es kann gewählt werden, ob sich die Kamera nach dem Austreten aus dem Trigger wieder auf ihren ursprünglichen Zoom ändert • Die Häufigkeit der Triggerüberprüfung ist anpassbar

Bis auf die Anforderung ein klar strukturiertes, einfaches und übersichtliches Interface zu haben, von welchem aus sich sämtliche Subsysteme steuern lassen, wurden alle Ziele erreicht.

5.2 Kritische Wertung

Ziel der vorliegenden Arbeit war es, anhand der Entwicklung einer Software mit den Teilschritten Konzeption, Design und Prototypentwicklung einen Einblick in das Forschungsfeld zu gewinnen.

Aufbauend auf bisherigen Erkenntnissen wurden die Herausforderungen der Kameras in Sidescrollern erörtert und in ein Stufensystem überführt. Dieses Stufensystem dient dabei lediglich der Kategorisierung verschiedener Methoden, gibt aber keine Wertung ab. So wird auch in den Analysen der verschiedenen Spiele zwar auf die Auswirkungen der Kamera auf das jeweilige Spiel eingegangen, dabei wird jedoch auf eine detaillierte Bewertung verzichtet. Um die tatsächlichen Auswirkungen verschiedener Arten der Kameraverfolgung auf unterschiedliche Spiele zu untersuchen, müsste eine empirische Erhebung mit quantitativem Forschungsansatz und standardisierten Verfahren durchgeführt werden.

Die Aufführung verschiedener Lösungsansätze für die einzelnen Komponenten erhebt nicht den Anspruch auf Vollständigkeit. Bedingt durch die begrenzte Kapazität wurde sich hier auf die Darstellung von zwei Lösungsansätzen pro Problem konzentriert. Vor allem im Bereich der Methoden zur Kameraverfolgung gibt es einige weitere interessante Ansätze (siehe z.B. Kapitel 2.4), welche jeweils verschiedene Vor- und Nachteile mit sich bringen.

Die ausgearbeitete Architektur des Kamerasystems ist soweit wie möglich allgemeingültig. So wurde beispielsweise auf die unityeigenen Collider verzichtet. Da sich für die Erstellung aber auf die Unity3D Engine konzentriert wurde, ist die

Vererbung von der unityeigenen Klasse *MonoBehaviour* nicht zu vermeiden. Bei der Umsetzung des Systems in andere Engines ist zu beachten, dass diese Klasse oder ein Äquivalent unter Umständen nicht vorhanden ist. Daraus resultierend entsteht die Gefahr, dass einige spezifische Methoden andersartig gelöst werden müssen, wenn das entwickelte Modell in andere Engines implementiert werden soll.

5.3 Ausblick

Wie bereits erwähnt, konnte bisher kein nutzerfreundliches Interface entwickelt werden. Um das System in Gebrauch zu nehmen, ist die Entwicklung eines solchen der nächste Schritt. Dann kann in anderen Arbeiten darauf aufgebaut werden. Besonders interessant scheint hier die Untersuchung des tatsächlichen Einflusses verschiedener Kameras auf ein Spiel.

Die in dieser Arbeit aufgezeigten Methoden zur Kameraverfolgung bilden nur einen Teil aller Möglichkeiten ab. Hinzu kommt, dass selbst in einem alten Genre wie den Sidescrollern immer noch neue Innovationen designt werden, wie das Beispiel „Mutant Mudds Deluxe“ zeigt. Daher kann ein einzelnes System niemals sämtliche Methoden abdecken. Vielmehr soll es eine Vereinfachung für kommende Innovationen bereitstellen. Das hier entwickelte Modell bietet einen Ansatz dazu. Wie diese Innovationen aussehen können lässt sich dabei nicht vorhersehen, doch genau dieser Umstand macht den Reiz der stetigen Weiterentwicklung aus.

“In game development, the first 90% of a project is a lot easier than the second 90 %”- Tim Sweeney, Gründer und Präsident von Epic Games und Entwickler der Unreal Engine[63]

[63] Sweeney, Tim (2015): *In game development, the first 90% of a project is a lot easier than the second 90%*, veröff. auf twitter.com, am 30.11.2015

Literaturverzeichnis

Balzert, Helmut (2011): *Lehrbuch der Softwaretechnik. Entwurf, Implementierung, Installation und Betrieb*, 3. Auflage, Heidelberg

Bogdan, Purcaru Ion (2014): *Games vs. Hardware. The History of PC Gaming. The 80's*, E-Book

Brandt-Pook, Hans / Kollmeier, Rainer (2008): *Softwareentwicklung kompakt und verständlich. Wie Softwaresysteme entstehen*, Wiesbaden

Bünte, Hermann (2015): *Vestibuläres System,* veröff. auf med-college.de, http://www.med-college.de/ru/wiki/artikel.php?id=1580&lan=1 [Zugriff am 09.12.2015]

Capcom AG (Hrsg.): *Mega Man Zero Official Complete Works,* 2008

CarlEmail (2012): *How to smooth damp towards a moving target without causing jitter in the movement?,* veröff. auf forum.unity3d.com, http://forum.unity3d.com/threads/how-to-smooth-damp-towards-a-moving-target-without-causing-jitter-in-the-movement.130920/#post-885121 [Zugriff am 09.12.2015]

CBS Interactive Inc. (Hrsg.) (2015): *Ori and the Blind Forest*, veröff. auf metacritic.com, http://www.metacritic.com/game/pc/ori-and-the-blind-forest [Zugriff am 09.12.2015]

Csuk, Maximilian (2012): *Cameras in 2D platformers*, veröff. auf imake-games.com, http://www.imake-games.com/cameras-in-2d-platformers/ [Zugriff am 09.12.2015]

Felzmann, Sebastian (2012): *Playing Yesterday - Mediennostalgie im Computerspiel*, Boizenburg

Fenty, Sean (2008): *Why Old School is ‚Cool'. A Brief Analysis of Classic Video Game Nostalgia*, in: Playing the Past : History and Nostalgia in Video Games, hg von Zach Whalen und Laurie N. Taylor, Nashville

Future US, Inc. (Hrsg.) (2015): *Gaming's most important evolutions. Scrolling*, veröff. auf gamesradar.com, http://www.gamesradar.com/gamings-most-important-evolutions/?page=2 [Zugriff am 09.12.2015]

Future US, Inc. (Hrsg.) (2015): *Gaming's most important evolutions. Parallax scrolling*, veröff. auf gamesradar.com, http://www.gamesradar.com/gamings-most-important-evolutions/?page=3 [Zugriff am 09.12.2015]

Gamer Network (Hrsg.) (2008): *The Tao of Beat-'em'-ups. Part 1: The evolution of a genre*, 1976 to 1985, Online-Artikel, veröff. auf eurogamer.net, http://www.eurogamer.net/articles/the-tao-of-beat-em-ups-article?page=2 [Zugriff am 09.12.2015]

Gilbert, Ron/Double Fine Productions, Inc. (2013): *The Cave*, veröff. auf thecavegame.com, http://thecavegame.com/about.html [Zugriff am 09.12.2015]

Haines, Eric (2014): *ArrayPrefs2*, veröff. auf wiki.unity3d.com, http://wiki.unity3d.com/index.php/ArrayPrefs2 [Zugriff am 09.12.2015]

Hallmann, Matthias (1990): *Prototyping komplexer Sofwaresysteme*, Wiesbaden

Hoss, Brian (2014): *Once Secret, Now Known: 'Ori and the Blind Forest' For the Xbox One Shined Brightly at E3*, Online-Artikel, veröff. auf highdefdigest.com, http://www.highdefdigest.com/news/show/games/ori-and-the-blind-forest/moon-studios/Microsoft/Xbox_One/once-secret-now-known-ori-and-the-blind-forest-for-the-xbox-one-shined-brightly-at-e3/15979 [Zugriff am 09.12.2015]

ITWissen (Hrsg.) (2015): *Benutzeroberfläche*, veröff. auf itwissen.info, http://www.itwissen.info/definition/lexikon/Benutzeroberflaeche-UI-user-interface.html [Zugriff am 09.12.2015]

Janalta Interactive Inc. (Hrsg.) (2015): *Scrolling*, veröff. auf techopedia.org, https://www.techopedia.com/definition/5469/scrolling [Zugriff am 09.12.2015]

Janalta Interactive Inc. (Hrsg.) (2015): *Side Scroller*, veröff. auf techopedia.org, https://www.techopedia.com/definition/27153/side-scroller [Zugriff am 09.12.2015]

Johner, Christian (2015): *ISO 9126 und ISO 25010*, veröff. auf johner-institut.de, https://www.johner-institut.de/blog/iec-62304-medizinische-software/iso-9126-und-iso-25010/ [Zugriff am 09.12.2015]

Kent, Steven (2001): *The Ultimate History of Video Games*. *From Pong to Pokémon and Beyond – The Story Behind the Craze That Touched Our Lives and Changed the World,* New York

Keren, Itay (2015): *Scroll Back: The Theory and Practice of Cameras in Side-Scrollers,* veröff. auf gamasutra.com, http://gamasutra.com/blogs/ItayKeren/20150511/243083/Scroll_Back_The_Theory_and_Practice_of_Cameras_in_SideScrollers.php [Zugriff am 09.12.2015]

Kesson, Malcolm (2002): *Using smoothstep*, veröff. auf fundza.com, http://www.fundza.com/rman_shaders/smoothstep/index.html [Zugriff am 09.12.2015]

McConville, Ciaran (2015): *Point-and-Click Adventure Games Are Dead*, veröff. auf evansreview.com, http://evansreview.com/science-technology/point-and-click-adventure-games-are-dead/11677 [Zugriff am 09.12.2015]

Moon Studios (Hrsg.) (2015): *Ori and the Blind Forest*, veröff. auf oriandtheblindforest.com, http://www.oriblindforest.com/#!// [Zugriff am 09.12.2015]

Parish, Jeremy (2009): *Metroidvania: Rekindling a Love Affair with the Old and the New,* veröff. auf 1up.com, http://www.1up.com/do/blogEntry?bId=8999257&publicUserId=5379721 [Zugriff am 09.12.2015]

Renegade Kid LLC (Hrsg.) (2012): *Mutant Mudds Deluxe*, veröff. auf renegadekid.com, http://www.renegadekid.com/mutantmudds.htm [Zugriff am 09.12.2015]

Rogers, Scott (2010): *Level Up! – The Guide to Great Video Game Design*, 1. Aufl., Chichester

Schnauder, Volker / Jarosch, Helmut / Thieme, Ilja (2011): *Praxis der Software-Entwicklung. Techniken - Instrumente - Methoden*, Renningen-Malmsheim

Semmler, Jan (2012): *Vorteile eines gut designten grafischen User-Interfaces*, veröff. auf jansemmler.de, http://www.jansemler.de/interface/vorteile-eines-gut-designten-grafischen-user-interfaces/ [Zugriff am 09.12.2015]

Sonofrage (2014): *Working with the coordinate system and game screen in Unity 2DK,* veröff. auf stackoverflow.com, http://stackoverflow.com/questions/21937544/working-with-the-coordinate-system-and-game-screen-in-unity-2d [Zugriff am 09.12.20015]

Stencyl, LLC (Hrsg.) (2015): *The Camera*, veröff. auf stencyl.com, http://www.stencyl.com/help/view/the-camera/ [Zugriff am 09.12.2015]

Super Mario Wiki (Hrsg.) (2015): *Shigeru Miyamoto*, veröff. auf mariowiki.com, http://www.mariowiki.com/Shigeru_Miyamoto#Quotes [Zugriff am 09.12.2015]

Sweeney, Tim (2015): *In game development, the first 90% of a project is a lot easier than the second 90%,* veröff. auf twitter.com, am 30.11.2015

The Monkey Island SCUMM Bar (Hrsg.) (2004): *The Secret of Creating Monkey Island - An Interview With Ron Gilbert, excerpt from LucasFilm Adventurer vol. 1, number 1, Fall 1990,* veröff. auf scummbar.com, http://scummbar.com/resources/articles/index.php?newssniffer=readarticle&article=1033 [Zugriff am 09.12.2015]

Unity Technologies (Hrsg.) (2015): *Vector3.Lerp,* veröff. auf docs.unity3d.com, http://docs.unity3d.com/ScriptReference/Vector3.Lerp.html [Zugriff am 09.12.2015]

Unity Technologies (Hrsg.) (2015): *Layers,* veröff. auf docs.unity3d.com, http://docs.unity3d.com/Manual/Layers.html [Zugriff am 09.12.2015]

Unity Technologies (Hrsg.) (2015): *Execution Order,* veröff. auf docs.unity3d.com, http://docs.unity3d.com/Manual/ExecutionOrder.html [Zugriff am 09.12.2015]

Unity rechnologies (Hrsg.) (2015): *System Requirements for Unity 5.3*, veröff. auf unity3d.com, https://unity3d.com/unity/system-requirements [Zugriff am 09.12.2015]

Upper One Games LLC (Hrsg.) (2014): *Never Alone (Kisima Ingitchuna)*, veröff. auf neveralonegame.com, http://neveralonegame.com/game/ [Zugriff am 09.12.2015]

Utter, Robert (2014): *How to Lerp like a pro*, veröff. auf chicounity3d.wordpress.com, https://chicounity3d.wordpress.com/2014/05/23/how-to-lerp-like-a-pro/ [Zugriff am 09.12.2015]

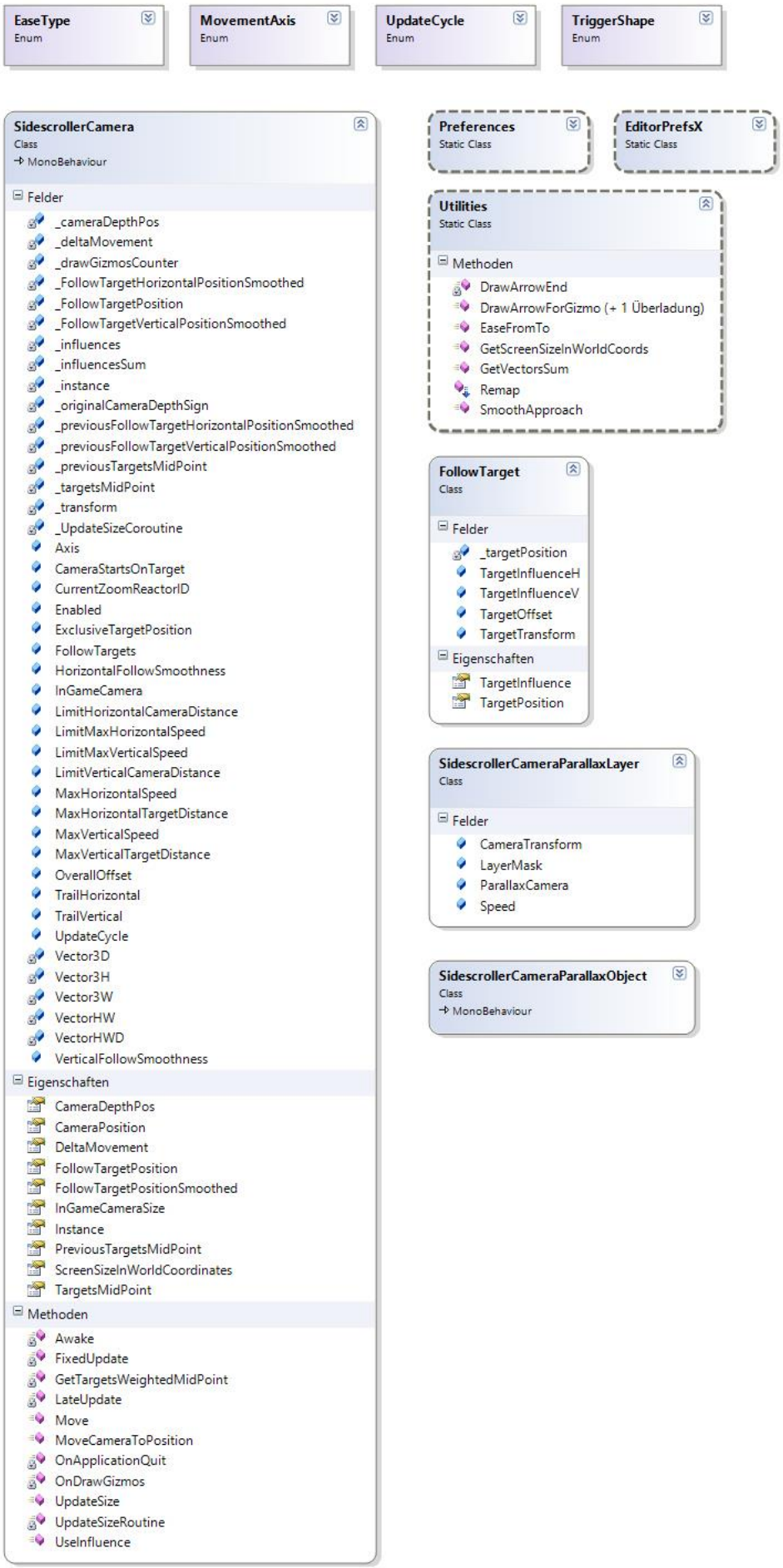
Yacht Club Games (Hrsg.) (2014): *Shovel Knight*, veröff. auf kickstarter.com, https://www.kickstarter.com/projects/yachtclubgames/shovel-knight/updates [Zugriff am 09.12.2015]

Yu, Derek (2008): *Spelunky v1.1 (and Source),* veröff. auf tigsource.com, https://forums.tigsource.com/index.php?topic=4017 [Zugriff am 09.12.2015]

Ziff Davis, LLC (Hrsg.) (2008): *The Leif Ericson Awards. The Long Jump*, Online-Artikel, veröff. auf ign.com, http://www.ign.com/articles/2008/03/24/the-leif-ericson-awards?page=2 [Zugriff am 09.12.2015]

Anlagen

Anlage 1: Klassendiagramm des Systems Sidescrollercamera


EaseType
Enum
MovementAxis
Enum
UpdateCycle
Enum
TriggerShape
Enum
SidescrollerCamera
Class
MonoBehaviour
Felder
_cameraDepthPos
_deltaMovement
_drawGizmosCounter
_FollowTargetHorizontalPositionSmoothed
_FollowTargetPosition
_FollowTargetVerticalPositionSmoothed
_influences
_influencesSum
_instance
_originalCameraDepthSign
_previousFollowTargetHorizontalPositionSmoothed
_previousFollowTargetVerticalPositionSmoothed
_previousTargetsMidPoint
_targetsMidPoint
_transform
_UpdateSizeCoroutine
Axis
CameraStartsOnTarget
CurrentZoomReactorID
Enabled
ExclusiveTargetPosition
FollowTargets
HorizontalFollowSmoothness
InGameCamera
LimitHorizontalCameraDistance
LimitMaxHorizontalSpeed
LimitMaxVerticalSpeed
LimitVerticalCameraDistance
MaxHorizontalSpeed
MaxHorizontalTargetDistance
MaxVerticalSpeed
MaxVerticalTargetDistance
OverallOffset
TrailHorizontal
TrailVertical
UpdateCycle
Vector3D
Vector3H
Vector3W
VectorHW
VectorHWD
VerticalFollowSmoothness
Eigenschaften
CameraDepthPos
CameraPosition
DeltaMovement
FollowTargetPosition
FollowTargetPositionSmoothed
InGameCameraSize
Instance
PreviousTargetsMidPoint
ScreenSizeInWorldCoordinates
TargetsMidPoint
Methoden
Awake
FixedUpdate
GetTargetsWeightedMidPoint
LateUpdate
Move
MoveCameraToPosition
OnApplicationQuit
OnDrawGizmos
UpdateSize
UpdateSizeRoutine
UseInfluence
Preferences
Static Class
EditorPrefsX
Static Class
Utilities
Static Class
Methoden
DrawArrowEnd
DrawArrowForGizmo (+ 1 Überladung)
EaseFromTo
GetScreenSizeInWorldCoords
GetVectorsSum
Remap
SmoothApproach
FollowTarget
Class
Felder
_targetPosition
TargetInfluenceH
TargetInfluenceV
TargetOffset
TargetTransform
Eigenschaften
TargetInfluence
TargetPosition
SidescrollerCameraParallaxLayer
Class
Felder
CameraTransform
LayerMask
ParallaxCamera
Speed
SidescrollerCameraParallaxObject
Class
MonoBehaviour

Anlage 2: Nutzerleitfaden zum System SidescrollerCamera

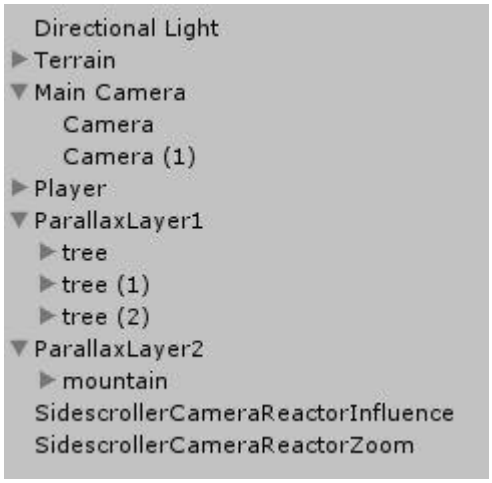
Installation

Um das System SidescrollerCamera als Plugin in der Unity3D Engine einzubinden, muss der Ordner „SidescrollerCamera“ mitsamt aller seiner Unterordner und Inhalte in den „Asset“-Ordner von Unity kopiert werden. Unity kompiliert daraufhin sämtliche Scripts, was einige Momente dauern kann.

Deinstallation

Um das Plugin „Sidescrollercamera“ aus dem Unity-Projekt zu entfernen, muss der Ordner „SidescrollerCamera“ mitsamt aller seiner Unterordner und Inhalte aus dem „Asset“-Ordner von Unity gelöscht werden. Eventuell erstellte Game Objects für Reactors müssen separat aus der Szene des jeweiligen Projekts gelöscht werden.

Aufbau einer Szene



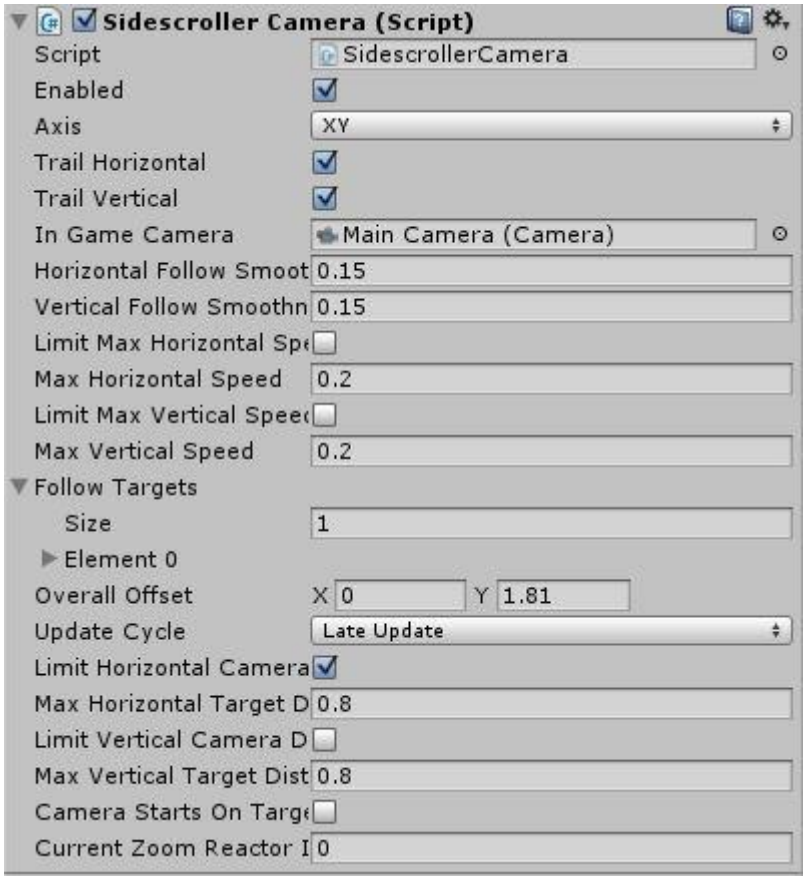
Beispiel eines Szenenaufbaus

Die Kameras für den Parallaxeffekt sind Kinder der Main Camera.

Sämtliche Objekte eines Parallaxlayers wurden unter einem Game Object zusammengefasst.

Die Reaktoren sind einzelne Game Objects.

Einstellung der Kameraverfolgung



Dies ist das Hauptinterface für die Kameraverfolgung. Das Script SidescrollerCamera.cs muss an die Main Camera angehängt werde. Daraufhin ist das Interface im Inspector View der Kamera erreichbar.

Einstelloptionen:

Axis:	Die Achsen auf der die Kamera sich bewegen soll
Trail Horizontal:	Aktivierung der horizontalen Kameraverfolgung
Trail Vertical:	Aktivierung der verticalen Kameraverfolgung
Horizontal Follow Smoothness:	Horizontale Sanftheit der Kamerabewegungen
Vertical Follow Smoothness:	Vertikale Sanftheit der Kamerabewegungen
Limit Max Horizontal Speed:	Aktivierung der horizontalen Geschwindigkeitsgrenze
Max Horizontal Speed:	Horizontale Höchstgeschwindigkeit
Limit Max Vertical Speed:	Aktivierung der vertikalen Geschwindigkeitsgrenze
Max Vertical Speed:	Vertikale Höchstgeschwindigkeit

Follow Targets:

Size:	Anzahl der Kameraverfolgungsziele
Element 0:	Erstes Kameraverfolgungsziel (Das Game Object, welches verfolgt werden soll, muss aus dem Scene View hier hereingezogen werden)
Overall Offset:	Verschiebung des Kameraverfolgungszieles vom eigentlichen Game Object weg
Update Cycle:	Wahl des Update Cycles, in welchem die Bewegung der Kamera stattfinden soll

Limit Horizontal Camera Distance:

Aktivierung der maximalen horizontalen Distanz zwischen Kamera und Verfolgungsziel

Max Horizontal Camera Distance:

Maximale horizontale Distanz zwischen Kamera und Verfolgungsziel

Limit Vertical Camera Distance:

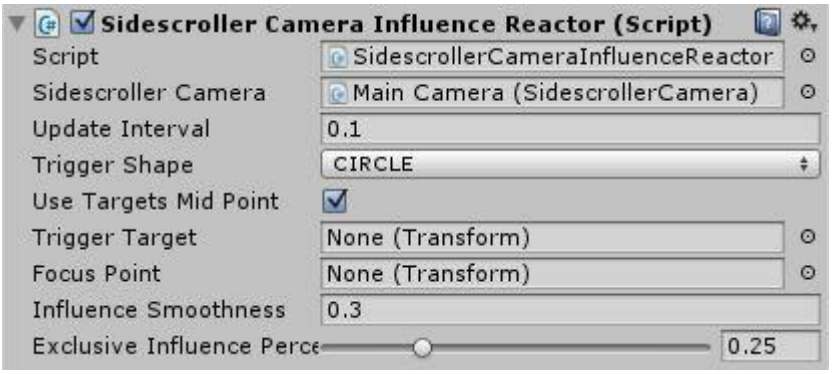
Aktivierung der maximalen vertikale Distanz zwischen Kamera und Verfolgungsziel

Max Vertical Camera Distance:

Maximale vertikale Distanz zwischen Kamera und Verfolgungsziel

Camera Starts On target: Wenn aktiviert zentriert sich die Kamera zu Beginn des Spiels sofort auf das Verfolgungsziel

Influence Reactors

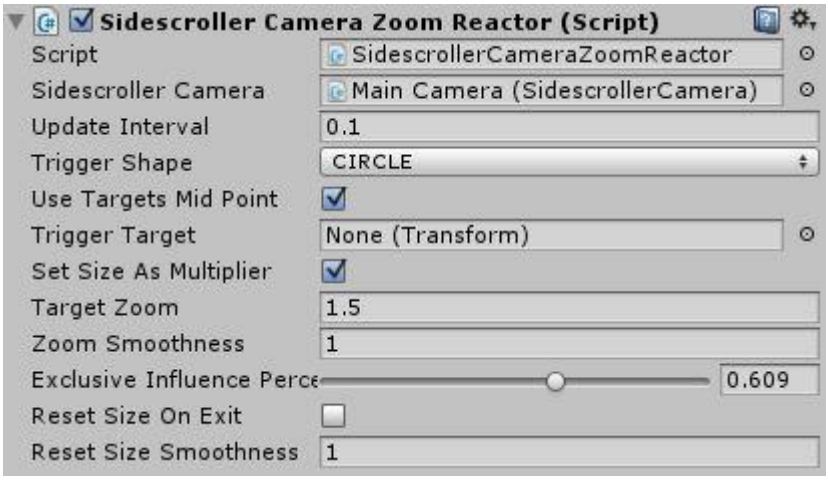


Um einen Einflussreaktor zu erstellen muss ein neues Game Object erstellt werden, welches das Script SidescrollerCameraInfluenceReactor.cs angehängt bekommt. Die Größe des Triggers wird über die Scale Einstellungen des Game Objects vorgenommen.

Einstelloptionen

Update Intervall:	Häufigkeit der Überprüfung ob der Spieler im Trigger ist
Trigger Shape:	Wahl zwischen rechteckigem oder kreisförmigem Trigger
Use Targets Mid Point:	Bei Aktivierung wird der Mittelpunkt des Spielers für die Triggerüberprüfung genommen, anstelle seiner Collider
Focus Point:	Hier muss ein Game Object eingefügt werden, wenn dieses für den Kamerafokus als finaler Fokuspunkt genutzt werden soll, anstelle des Mittelpunktes des Triggers
Influence Smoothness:	Sanftheit des Einflusses durch den Reaktor
Exclusive Influence Percentage:	Angabe zu welchen Teilen der absolute Triggerbereich den Trigger ausfüllen soll

Zoom Reactors



Um einen Zoomreaktor zu erstellen muss ein neues Game Object erstellt werden, welches das Script SidescrollerCamerazoomReactor.cs angehängt bekommt. Die Größe des Triggers wird über die Scale Einstellungen des Game Objects vorgenommen.

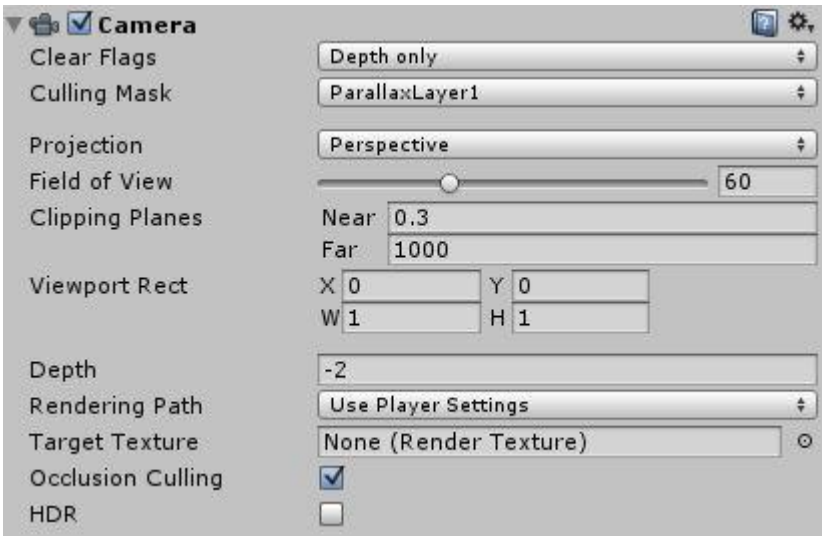
Eintelloptionen

Update Intervall:	Häufigkeit der Überprüfung ob der Spieler im Trigger ist
Trigger Shape:	Wahl zwischen rechteckigem oder kreisförmigem Trigger
Use Targets Mid Point:	Bei Aktivierung wird der Mittelpunkt des Spielers für die Triggerüberprüfung genommen, anstelle seiner Collider
Set Size as Multiplier:	Wenn aktiviert wird der Zoomwert als Multiplikator für den aktuellen Zoom genommen
Target Zoom:	Größenveränderung des Kameraausschnittes
Zoom Smoothness:	Sanftheit des Zooms
Exclusive Influence Percentage:	Angabe zu welchen Teilen der absolute Triggerbereich den Trigger ausfüllen soll
Reset Size on Exit:	Wenn aktiviert wird der Kameraausschnitt von vor dem Betreten des Triggers nach dem Austritt aus den Trigger widerhergestellt
Reset Size Smoothness:	Sanftheit der Größenwiderherstellung

Parallaxeffekt

Um einen Parallaxeffekt zu erhalten muss das Script SidescrollerCameraParallax.cs an die Main Camera angehängt werden. Des Weiteren muss für jede geplante Parallaxebene eine Kamera angelegt werden, welche ein Kind der Main Camera ist.

Die Einstellungen der der Parallaxkameras muss aussehen wie in diesem Bild: (Die Culling Mask muss auf das jeweilige Parallaxlayer eingestellt sein)



Einstelloptionen

Parallax Layers:

Size:	Menge der Parallaxebenen
Element 0:	Erste Parallaxkamera
Parallaxcamera:	An dieser Stelle muss die Parallaxkamera aus dem Scene View eingefügt werden
Speed:	Geschwindigkeit der Parallaxbewegung
Layer Mask:	Hier muss das gewünschte Parallaxlayer ausgewählt werden

Parallax Horizontal:	Aktivierung der horizontalen Parallaxbewegung
Parallax Vertical:	Aktivierung der vertikalen Parallaxbewegung

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe. Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Ort, Datum Vorname Nachname